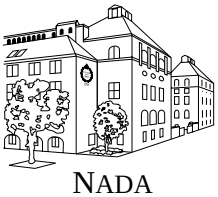

NADA
Numerisk analys och datalogi
Kungl Tekniska Högskolan
100 44 STOCKHOLM

Department of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, SWEDEN

Cryptanalysis of Symmetric Block Ciphers – Breaking Reduced KHAZAD *and* SAFER++

Kryptoanalys av symmetriska blockkrypton – Knäckning av försvagad KHAZAD *och* SAFER++

Gustaf Dellkrantz
(d98-gde@nada.kth.se)

TRITA-NA-Eyynn

Master's Thesis in Computer Science (20 credits)
at the School of Computer Science and Engineering,
Royal Institute of Technology, August 2003
Supervisor at Nada was Johan Håstad
Examiner was Johan Håstad
Work performed at K.U. Leuven, Belgium
Supervisor at K.U. Leuven was Christophe De Cannière

Cryptanalysis of Symmetric Block Ciphers

Breaking Reduced KHAZAD and SAFER++

Abstract

This thesis is focused on the cryptanalysis of block ciphers. We give an introduction to the theory of block ciphers and an overview of cryptanalytic attacks. Our contribution is the analysis of two ciphers, KHAZAD and SAFER++.

We exploit the simple mathematical structure of a version of KHAZAD reduced from 8 to 5 rounds and show the existence of 2^{64} weak keys that can be broken with 2^{32} chosen plaintexts, 2^{33} adaptively chosen ciphertexts and 2^{40} steps of computation. The weak keys are a consequence of the involutional structure and can be exploited using a slide with a twist attack. Our results do not threaten the full cipher, but suggest that more analysis is necessary with respect to the simple mathematical structure.

Our attack on SAFER++ breaks 5.5 out of 7 rounds. We show a boomerang attack using truncated differentials and a novel middle round trick, based on the mathematical structure of the S-boxes used in SAFER++. For 5 rounds, the complexity of the attack is 2^{77} chosen plaintexts, 2^{77} adaptively chosen ciphertexts and work approximately equal to 2^{78} encryptions. Extending the attack to 5.5 rounds results in an attack needing 2^{107} chosen plaintexts, 2^{107} adaptively chosen ciphertexts and work approximately equal to 2^{108} encryptions.

Kryptoanalys av symmetriska blockkrypton

Knäckning av försvagad KHAZAD och SAFER++

Sammanfattning

Detta examensarbete behandlar analys av blockkrypton. Det innehåller en introduktion till blockkrypton samt en sammanställning av några kryptografiska attacker. Vårt bidrag är analys av två krypton, KHAZAD och SAFER++.

Den enkla matematiska strukturen hos KHAZAD analyseras. Vi visar att KHAZAD reducerad från 8 till 5 varv har 2^{64} svaga nycklar. Dessa nycklar kan knäckas med 2^{32} valda klartexter, 2^{33} adaptivt valda kryptotexter och 2^{40} beräkningssteg. De svaga nycklarna uppstår på grund av kryptots struktur, där alla komponenter är involutioner, och knäcks med en "slide with a twist"-attack. Detta hotar inte KHAZADs säkerhet, men visar att den matematiska strukturen borde analyseras närmare.

Vi attackerar SAFER++ reducerad från 7 till 5,5 varv. Attacken är en bumerangattack som använder trunkerade differentialer och de matematiska egenskaperna hos de ickelinjära funktioner som SAFER++ baseras på. En attack mot 5 varv kräver 2^{77} valda klartexter, 2^{77} adaptivt valda kryptotexter och tid motsvarande kryptering av 2^{78} block. Attacken kan utökas till 5,5 varv och kräver då 2^{107} valda klartexter, 2^{107} adaptivt valda kryptotexter och tid motsvarande 2^{108} krypteringar.

Preface

The work presented in this thesis was performed at the COSIC research group of the ESAT department of the Katholieke Universiteit Leuven, Belgium, between August 2002 and March 2003. It is a master's project at the School of Computer Science and Engineering of the Royal Institute of Technology (KTH), Sweden.

I would like to thank the people that helped me during the work. At KTH, my supervisor, professor Johan Håstad, provided me with comments and encouragement from before the start of my work to the end.

At COSIC, I would like to thank my supervisor, Christophe De Cannière, for ideas, comments, encouragement, taking care of practical matters and for his contributions to the results presented in this thesis. Furthermore, many thanks to professor Alex Biryukov for ideas and comments, as well as for his contributions to the results presented here. Thanks also to Jorge Nakahara, Jr. for some ideas and introduction to the family of SAFER-ciphers and to professor Bart Preneel for making my stay at COSIC possible.

Contents

List of Figures

Chapter 1

Introduction

1.1 Cryptography

This thesis is focused on the subject of cryptanalysis. Cryptanalysis is the science of breaking ciphers. The cryptanalyst must often use techniques from several different subjects, ranging from computer science over algebra to probability theory. The beauty is in the artistic nature of the cryptanalysis, when the cryptanalyst tries to find and exploit the mistakes or oversights of the cryptographer who designed the cipher.

We study symmetric cryptography which can be used to secure communication between two parties who have beforehand agreed on a secret key to use. Asymmetric cryptography, sometimes referred to as public-key cryptography, allows encrypted communication without the need for agreeing on a secret key beforehand. As symmetric encryption algorithms usually are many times faster than asymmetric ones a common way of use is to combine them. That is, asymmetric cryptography is used to establish a shared secret key which is then used in a symmetric encryption algorithm to encrypt the actual data.

The need for strong cryptography is ever rising. E-business, money transfers between banks and confidentiality of digitally stored medical records are three examples of situations where strong cryptography is needed for security. Governments and the military are other users of cryptography, and have been so for many decades. Research in cryptography has mostly been performed in the closed quarters of military agencies, but in the last 30 years the research in the open academic community has increased heavily.

Some terminology (see Fig. 1.1): the message to be encrypted is called the *plaintext*, the encrypted message is called the *ciphertext*, the encryption algorithm is a *cipher* and the act of transforming the message into an unreadable form is *encrypting* or *enciphering*. Recovering the plaintext from the ciphertext using the key is called *decrypting* or *deciphering*.

Symmetric ciphers can be divided into two classes, stream and block ciphers. Stream ciphers have internal state while block ciphers are stateless. This distinction

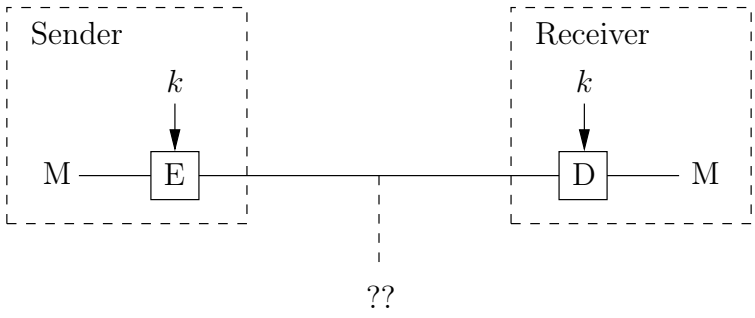

Figure 1.1. An example of an encrypted communications setting. The plaintext M is sent encrypted using the encryption algorithm E and key k and then decrypted using the decryption algorithm D. An eavesdropper not knowing the key k should not be able to gain any information on the plaintext sent

is not strict as stream ciphers can be built from block ciphers and vice versa.

This thesis focuses on the analysis of block ciphers. One of the goals of a block cipher is that it should not be possible to recover information on the key or plaintext from ciphertexts or even given unlimited access to an encryption engine with an embedded key. Cryptanalysis is the subject of breaking ciphers and our work is focused on two ciphers, called KHAZAD and SAFER++ respectively. We will devise attacks on reduced versions and ultimately try to show that the ciphers do not meet the requirements for block ciphers.

For an introduction to cryptography containing a large number of algorithms, protocols and further references, see [54]. A more technical and recent work is [41]. Building a secure and dependable system requires much more than cryptography and there are many pitfalls. For an excellent book on this subject, see [1].

1.2 Outline of Thesis

This thesis is organized as follows. This chapter, Chapter 1, contains an brief introduction, this outline and a short summary of the results presented in the thesis.

In Chapter 2 the general theory of block ciphers is introduced. A definition of the concept of block ciphers is given together with some discussion on their use and construction. The chapter ends with a classification of attacks and an overview of attacks on block ciphers from the literature.

Chapter 3 contains our analysis of the block cipher KHAZAD. A description of the cipher is given together with the best previous attacks, followed by our analysis using slide with a twist techniques to find weak keys.

In Chapter 4 the block cipher SAFER++ is analyzed. After a description of the cipher and a summary of the previous analysis done, some properties of the cipher are investigated. These properties are then used in a boomerang attack which significantly improves the best known results for this cipher.

The thesis ends with conclusions and some discussion on future work, given in Chapter 5.

1.3 Summary of Results

In this section we give a short summary of the results presented in this thesis.

1.3.1 Khazad

The analysis of the block cipher KHAZAD in Chapter 3 focuses on a version reduced from 8 to 5 rounds. The block size of the cipher is 64 bits and the key length is 128 bits. The best previous result is an attack on 4 rounds using 2^9 chosen plaintexts and 2^{80} steps of computation.

Utilizing the involutional structure of the cipher we show the existence of weak keys in 5 round KHAZAD. We use slide with a twist techniques and show that there are 2^{64} keys that can be broken with 2^{32} chosen plaintexts, 2^{33} adaptively chosen ciphertexts and 2^{40} steps of computation. The success probability of the attack is approximately 0.63.

The set of weak keys can be extended using techniques from differential cryptanalysis.

1.3.2 SAFER++

In Chapter 4 the 128-bit block and 128-bit key version of SAFER++ is analyzed. The cipher is the latest cipher in a family of substitution permutation networks. The best previous attack is a linear attack on 3 rounds which works for a fraction 2^{-13} of keys. The complexity of the attack is 2^{81} known plaintexts and 2^{101} steps of computation.

We show a boomerang attack using truncated differentials on SAFER++ reduced from 7 to 5.5 rounds. The attack relies on a novel middle round trick using the mathematical structure of the S-boxes. The complexity of the attack on 5 rounds is 2^{77} chosen plaintexts, 2^{77} adaptively chosen ciphertexts and time approximately equivalent to 2^{78} encryptions, with a success probability of 0.90.

The 5 round attack can be extended to breaking 5.5 rounds by guessing 30 bits of the key and then performing the previous attack, giving a complexity of 2^{107} chosen plaintexts, 2^{107} adaptively chosen ciphertexts and time approximately equivalent to 2^{108} encryptions.

Chapter 2

Block Ciphers

Block ciphers are fundamental building blocks of cryptographic protocols. This chapter contains a definition of block ciphers, some introduction to their use and construction and some discussion of what it means for a block cipher to be broken. The chapter ends with a description of some attacks that have appeared in the literature.

For more information on the subject, see [41], [54], references therein and the references given in this chapter.

2.1 Definition of Block Cipher

A block cipher is a function from a set of plaintexts and a set of keys to a set of ciphertexts. Early encryption algorithms, such as simple substitution and the Caesar cipher, can be viewed as block ciphers operating on single characters.

Formally, we can write a block cipher as a function

$$\mathrm{E} : \mathcal{K} \times \mathcal{P} \to \mathcal{C}$$

where $\mathcal{K}$ is the set of keys, $\mathcal{P}$ is the set of plaintexts and $\mathcal{C}$ is the set of ciphertexts.

Generally, block ciphers are specified as mathematical algorithms suited for computer implementation. They work on bit-sequences of some fixed length N. The key is also a bit-sequence of some fixed length K. It is therefore natural to consider block ciphers as functions

$$\mathrm{E} : \{0,1\}^K \times \{0,1\}^N \to \{0,1\}^N$$

where K is the key size and N is the block size of the cipher. Common values are $K \in \{64, 128, 192, 256\}$ and $N \in \{64, 128\}$. Note that a key of 64 bits is considered too small as it is likely to be breakable by exhaustive search.

Since it is necessary to be able to recover the plaintext from the ciphertext using the key, we require that for all keys $k \in \mathcal{K}$ the function E_k, which is encryption using key k, is a permutation. That is,

$$\forall k \in \mathcal{K},\ \mathrm{E}_k : \{0,1\}^N \to \{0,1\}^N$$

is a bijection. We denote encryption of message M under key k yielding ciphertext C by

$$\mathrm{E}_k(M) = \mathrm{E}(k, M) = C$$

and similarly, decryption of C under key k is denoted by

$$\mathrm{D}_k(C) = \mathrm{D}(k, C) = M$$

2.2 Why Block Ciphers?

2.2.1 Confidentiality

One major use for block ciphers is providing confidentiality, that is, preventing eavesdroppers from learning the contents of a message. A block cipher is a transformation on a fixed-size block and since the size of messages varies the need for modes of operation arises.

The most basic mode of operation for block ciphers is Electronic Codebook (ECB) mode, so named since it divides the message into N-bit blocks and encrypts the blocks individually. This mode has the drawback that equal blocks in different positions in the plaintext encrypt to equal ciphertext blocks, allowing an eavesdropper to learn some information on the plaintext.

Cipher Block Chaining (CBC) is another block cipher mode of operation that overcomes some of the obvious problems with Electronic Codebook mode. Cipher Block Chaining divides the message into N-bit blocks, but the blocks are chained so that the encryption of one block affects the encryptions of all subsequent blocks.

More information on block cipher modes of operations can be found in [41] and [47].

2.2.2 Building-Block

Block ciphers are not only useful for providing confidentiality using such a mode of operation. Some of the various ways of using a block cipher for other purposes are mentioned here. More information can be found in [41].

Stream ciphers can be built from block ciphers. Several modes of operations for block ciphers can be viewed as stream ciphers. Examples include Cipher Feedback (CFB) mode, Output Feedback (OFB) mode and Counter (CTR) mode.

Hash functions are useful in digital signature protocols and in various constructions such as HMAC [48] for providing message integrity. Hash functions can be built from block ciphers, using modes such as Miyaguchi-Preneel or Davies-Meyer.

Message Authentication Codes (MAC) provides message integrity and authentication. MACs can be built from hash functions, but also from block ciphers.

Pseudorandom Number Generators (PRNG) can be used to generate initialization vectors and session keys in cryptographic protocols. Block ciphers are suitable for such purposes, for example in Output Feedback (OFB) or Counter (CTR) mode.

2.3 Constructing Block Ciphers

This section contains a short introduction to some aspects of the inner workings of block ciphers.

The design of block ciphers goes back to Shannon [55] who introduced the notions of *confusion* and *diffusion* that are still used.

Confusion serves to make the relation between the plaintext, key and ciphertext complicated and hard to analyze. A simple way of doing this is by *substitution*, that is exchanging some block of bits for another, for example by table lookup. To frustrate mathematical analysis, nonlinear functions are used in the substitution steps of block ciphers.

Diffusion serves to spread the redundancy and patterns of the plaintext over the ciphertext making them harder to exploit. One way of accomplishing this is by *transposition* (also known as *permutation*), that is rearranging the order of pieces of the plaintext. Transposition can be done on characters, blocks of bits or individual bits.

2.3.1 Product Ciphers

Block ciphers are often built by iterating simple, nonlinear functions a number of times. The justification is that several iterations of a simple nonlinear function can result in a complicated, nonlinear function while still being reasonably easy to analyse, describe and implement. One iteration is called a *round* and the function iterated is called the *round function*. It is common to use the same round function in all rounds, changing only the key that is used.

2.3.2 Substitution Permutation Networks (SPNs)

Following the ideas of Shannon on confusion and diffusion, many ciphers are built in a layered Substitution Permutation (SP) structure. Each layer has a special purpose:

The substitution layer applies a nonlinear substitution to the block under encryption. The nonlinear function is often implemented using a substitution table, commonly called an *S-box*. This step is done to ensure that the dependency between the plaintext, key and ciphertext is nonlinear.

The permutation layer mixes the different parts of a block, often using a linear function. Using more general linear or affine functions than simple transposition the diffusive power can be increased. This step ensures that changes in plaintexts and keys are spread throughout the block.

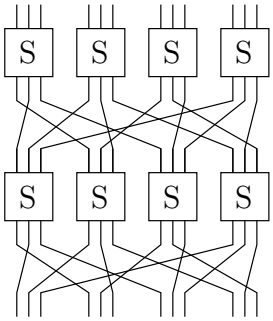

Figure 2.1. A two round iterated Substitution Permutation Network

The key mixing layer applies the key or a subkey to the block. This step mixes the key with the block making the relation between the plaintext and the ciphertext key dependent. The mixing of the key with the block is often performed using xor or modular addition.

As noted, there are no reasons not to use a more general linear function than a transposition in the permutation layer. Such a cipher can be called a Substitution Linear Network (SLN), but we will keep the more commonly used term substitution permutation network.

An example of a two round iterated substitution permutation network is shown in Fig. 2.1.

2.3.3 Feistel Ciphers

Feistel ciphers are a class of substitution permutation networks with a special structure. Invented by Horst Feistel [20], the general structure (as seen in Fig. 2.2) has been used in a number of block ciphers, such as DES [46] and Blowfish [53].

The input block is split into two halves, L_0 and R_0, that are then updated using some arbitrary function f and the recursion

$$\begin{aligned} R_i &= L_{i-1} \oplus \mathrm{f}(k_{i-1}, R_{i-1}) \\ L_i &= R_{i-1} \end{aligned}$$

where k_{i-1} is the subkey used. This construction is always invertible, even if f is not. If the swap is left out in the last round, decryption is equal to encryption with the subkeys k_i in the reverse order. This is an implementation advantage as the same piece of code can do both encryption and decryption with only a change in the indexing of subkeys.

2.3.4 Round Functions

As mentioned previously, block ciphers are often built by iterating relatively simple functions, called round functions, several times. It is common to build round func-

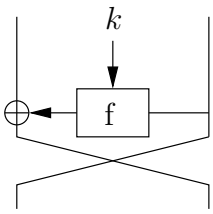

Figure 2.2. The Feistel structure

tions from two components, nonlinear functions and linear transformations. This section covers some of the ideas used in the design of round functions.

Nonlinear Functions

The nonlinear function of a block cipher is a main source of cryptographic strength and much research has been done in the area. It is common to use a small function in parallel on the input of the round function. The block might for example be split into bytes followed by applying some function to each byte individually. These small functions are often table-based and are then referred to as S-boxes. The size of an S-box is written $m \times n$, where m is the number of input bits and n is the number of output bits. Common sizes are 8×8, 6×4 and 8×32.

Three approaches to the design of nonlinear functions can be identified:

Randomly chosen S-boxes have been used in a number of ciphers. Some, such as Blowfish [53] use key dependent randomly chosen S-boxes.

Chosen and tested S-boxes have been used in several ciphers. One example is Khazad [3], which uses a fixed, randomly generated S-box that was tested for certain properties during the design of the cipher.

Mathematically generated S-boxes are constructed using some mathematical function. One example of this approach is in [51]. Ciphers using this type of S-boxes include Rijndael [19] and Safer++ [38].

Each of these approaches comes with advantages and disadvantages. Randomly chosen S-boxes usually lack structure that might help the cryptanalyst, but in the case of key dependently generated S-boxes there might be keys that generate weak S-boxes in some respect. The modular multiplication found in IDEA [32][33] can be seen as a key dependent randomly selected S-box and IDEA indeed has weak keys, as shown in [17], [23] and [11].

Mathematically generated S-boxes can be constructed to have a number of desirable properties with respect to attacks such as differential and linear cryptanalysis. One example is the S-box used in Rijndael that is based on inversion in a finite field. However, care must be taken to avoid attacks, such as the interpolation attack [24], that use the mathematical structure of the cipher. In Chapter 4 of this thesis the mathematical structure of the S-boxes of Safer++ is used to improve an attack.

Linear Transformations

The linear layer, or the linear transformation, of a block cipher serves to spread local changes in the block. These propagation properties have fundamental impact on the effectiveness of several cryptanalytic attacks, such as differential and linear cryptanalysis.

To measure the diffusive power of a linear transformation, the *branch number* [16] can be used.

Definition 2.1. The *Hamming weight* $\mathrm{w_h}$ of a vector is the number of components of the vector that are nonzero.

Definition 2.2 [16]. The *branch number* B of a linear transformation θ is

$$\mathrm{B}(\theta) = \min_{a \neq 0} \{\mathrm{w_h}(a) + \mathrm{w_h}(\theta(a))\}$$

The branch number of a linear transformation is thus the smallest possible sum of the number of active components of the input and output vectors of the linear transformation. A linear transformation on a vector space of dimension D can have a branch number as high as $D + 1$. Such a transform is called Maximum Distance Separable (MDS) and can be constructed, for example, using the theory of error-correcting codes.

Branch is often measured on the byte-level, viewing the linear transform as a transform on byte vectors, but can also be measured on the bit-level. Note that the bit-level branch of a transform is always strictly smaller than $D + 1$.

Bit-level permutations, used for example in DES, generally have a low branch number. This makes the resistance to differential and linear cryptanalysis hard to analyse and highly dependent on the actual choice of S-boxes.

More recent ciphers, such as Rijndael and Khazad, use linear transformations with known high branch. This allows the designers to show that certain attacks, such as linear and differential cryptanalysis, are unlikely to work.

2.3.5 Key Scheduling

The key schedule of a block cipher takes the master key, provided by the user, and calculates the actual subkeys used in the rounds of the cipher.

The purpose of the key scheduling is to spread the influence of individual key bits into the block under encryption through the subkeys. Less than optimal key schedules might lead to weak keys as in DES and IDEA [17][23][11], complementation properties as in DES or related key attacks [5] as the one on RC4 in [21].

Key scheduling can be performed in a number of different ways. DES uses a simple bit-permutation while Blowfish uses the cipher itself for key scheduling. A slow, complicated key schedule can help protect against attacks but might make the cipher unsuitable for use in situations where frequent key change is needed.

2.4 Breaking a Block Cipher

What it means to break a block cipher varies with the situation. It depends on the protocol using the cipher, the threat model and the power of the attacker. In a setting where the cipher is used to provide confidentiality, an efficient algorithm to recover the plaintext from the ciphertext without knowledge of the key is a break. If the cipher is used to build a hash function the cipher might be broken if there exists an efficient way to find two different keys that encrypt some fixed plaintext into two related ciphertexts.

In general a cipher is consider broken if there is some kind of weakness in the cipher that makes it perform worse than expected in some respect. This is captured in the terms *K-secure* and *hermetic*, as defined in [16] and restated here. Note that the definitions are not to be read as mathematical definitions, but rather as defining useful concepts.

Definition 2.3 [16]. A block cipher is *K-secure* if all possible attack strategies for it have the same expected work factor and storage requirements as for the majority of possible block ciphers with the same dimensions. This must be the case for all possible modes of access for the adversary and for any a priori key distribution.

Definition 2.4 [16]. A block cipher is *hermetic* if it does not have weaknesses that are not present for the majority of block ciphers with the same block and key length.

These definitions should be read in the context of block ciphers as randomly selected permutations from the set of all possible permutations. Essentially, the definitions mean that the block cipher in question works as good as we can expect it to. For example, it is possible to imagine a K-secure and hermetic block cipher with an 8-bit block. Such a cipher would fall to such a simple attack as frequency analysis, but so would all other possible block ciphers with an 8-bit block.

It is common to consider reduced variants of ciphers and to try and break them. Instead of analyzing the full cipher, the cryptanalyst might analyze versions with fewer rounds or weakened in other ways. This can provide some insight into the strength and security margin of the cipher, as a cipher where all but a few rounds are breakable might inspire less trust in the design than a cipher where not even a reduced version has been broken. Since it is currently unknown how to prove that a cipher is secure against all attacks, the strength of a cipher is usually estimated by the analysis performed of it and how easily broken reduced versions are.

2.4.1 Classification According to Result

One way of classifying attacks is according to what the result of a successful attack is. The ultimate goal of the attacker is usually to recover the secret key, but the attacker might settle for less in some situations. When doing an evaluation of a cipher, attacks that reveal a possibility for any of the following results are usually enough to conclude that the cipher is inadequate.

Distinguish from random is the attacker being able to distinguish the output of the cipher from the output of a randomly selected permutation. This is undesirable, for example, if the cipher is used as a pseudorandom number generator. A distinguishing attack is often the first step towards a more powerful attack.

Ciphertext forgery is the ability of the attacker to forge ciphertexts that decrypt to meaningful plaintexts. It might for example be the case that a change in the most significant bit of the plaintext block only affects the most significant bit of the ciphertext. Such a property is undesirable, but might not allow for plaintext or key recovery.

Plaintext recovery is the ability of the attacker to recover the plaintext corresponding to some ciphertext without knowledge of the key. A plaintext recovery attack does not necessarily allow the attacker to recover the whole plaintext, but even recovering a small piece of the plaintext is undesirable.

Key recovery is the possibility for the attacker to recover the secret key. This is the worst failure possible for a cipher as it allows the attacker to recover the plaintext, forge ciphertexts and distinguish the cipher from random.

2.4.2 Classification According to Power of Attacker

Another useful way of classifying cryptanalytic attacks is according to the power the attacker needs, to be able to carry them out. The following is one such classification:

Ciphertext only (CO) is an attack in which the attacker is assumed to have knowledge of some ciphertexts and a model of the distribution of the corresponding plaintexts. The attacker might for example be eavesdropping on an encrypted transmission knowing that the plaintext is ASCII-encoded English. This kind of attack is not very powerful, but can be applied to various old or simple ciphers.

Known plaintext (KP) gives the attacker access to a set of plaintexts and the corresponding ciphertexts. The attacker might for example eavesdrop on an encrypted transmission and then get the plaintext using some other means, such as breaking in or bribing someone. Known headers or a standard letter heading are other common sources of known plaintext.

Chosen plaintext (CP) attacks give the attacker the ciphertexts that correspond to plaintexts that were chosen by the attacker. This is a powerful class of attacks that a secure block cipher must resist. Chosen plaintext attacks are harder to carry out for the attacker, but can be done, for example, if an encryption box with an embedded secret key falls into the hands of the attacker.

Adaptive chosen plaintext (ACP) is similar to the chosen plaintext attack, but the attacker is allowed to choose the plaintexts after analyzing previously chosen plaintext/ciphertext pairs.

Chosen ciphertext (CC) is similar to a chosen plaintext attack, but the attacker chooses a set of ciphertexts and then gets the plaintexts that results from decrypting the chosen ciphertexts. This kind of attack might work better than a chosen plaintext attack if the encryption and decryption structures of the cipher differ significantly.

Adaptive chosen ciphertext (ACC) is similar to an adaptive chosen plaintext attack, but the attacker chooses the ciphertexts instead of the plaintexts.

Related key (RK) attacks give the attacker access to two (or more) encryption engines where the secret keys are unknown but related in some known way. The weakness [21] in RC4 as used in the Wired Equivalent Privacy (WEP) protocol is one example of a related key weakness.

2.4.3 Complexity of Attacks

The complexity of a cryptanalytic attack on a block cipher is measured using four different measures:

Work is a measure of the steps of computation the attacker needs to do to carry out the attack. Can be measured in equivalent number of encryptions or in number of simple operations.

Data is the amount of data needed for the attack and is measured in blocks. Depending on the type of attack it might be the number of chosen plaintext blocks or the amount of known plaintext required.

Storage is the amount of storage space the attacker needs to carry out the attack. Usually measured in blocks of memory required.

Probability of success is the probability that the attack will succeed, given some fixed amount of work, data and storage. There is often a tradeoff between probability of success and the other complexities of an attack.

Of the three resources work, data and storage, data is usually considered to be the most expensive since work and storage burdens the attacker exclusively while data requires the attacker to interact with the target. Getting the target to encrypt 2^{40} chosen plaintexts is usually significantly harder for the attacker than doing 2^{40} computations on the attackers own computers. As the complexity of exhaustive key search is 2^K for a cipher with a K-bit key, any attack must have time complexity significantly lower than 2^K encryptions to be worthwhile.

2.5 Attacks on Block Ciphers

In this section we give an overview of some attacks against block ciphers. The descriptions are not intended to cover all details, but are still quite technical due to the nature of the subject. More information can be found in the references given.

2.5.1 Exhaustive Key Search

Exhaustive key search is the attacker trying decryption of an intercepted ciphertext with every possible key until a known or likely plaintext is produced.

The attack can be performed either if the attacker has a few known plaintexts or if the plaintexts contain enough redundancy that a correct decryption can be identified, which is very often the case. To protect against this attack the number of possible keys must be large. For example, RIJNDAEL accepts a key between 128 and 256 bits in length, making exhaustive key search infeasible. DES on the other hand has a 56-bit key, which can be searched in a few days using dedicated hardware or a network of computers.

2.5.2 Key Recovery Using Distinguishers

A number of cryptanalytic attacks on block ciphers are based on distinguishers. The general idea for such an attack on an r-round cipher is to find a property of the intermediate value after less than r rounds. Assuming that the round function is somewhat random-looking with respect to this property, a correct guess of the subkeys used in the last rounds can be distinguished from an incorrect guess.

Assume that for a block cipher some property holds after $r - 1$ rounds with probability p and that for a random permutation this property holds with probability q. A correct guess of the subkey used in the last round can now be identified using the fact that in a set of M ciphertexts, partially decrypted using the guessed subkey, about pM texts should have the property. If the guess of the subkey is wrong, about qM texts should have the property.

Properties useful for these kinds of attacks include sums of the values in well-chosen sets, the difference between two values and the parity of some bits.

2.5.3 Differential Cryptanalysis

Differential cryptanalysis is a statistical chosen plaintext attack introduced by Biham and Shamir [9]. Briefly, it can be described as tracing the evolution of differences between the intermediate values of encryptions of two suitably related plaintexts through the rounds of a cipher. If the propagation of a fixed difference in the plaintexts to some fixed difference in the ciphertexts has high enough probability, the cipher can be distinguished from random.

Differential cryptanalysis is a general attack applicable to many block ciphers and was the first attack on full DES faster than exhaustive search [8].

The notion of difference used in differential cryptanalysis varies with the cipher to be attacked. It is common to use the inverse of the operation used to insert the subkeys as the difference operation. For example, if the subkeys are inserted using xor, a natural choice is to consider xor differences. This makes the differences somewhat independent of the unknown subkeys.

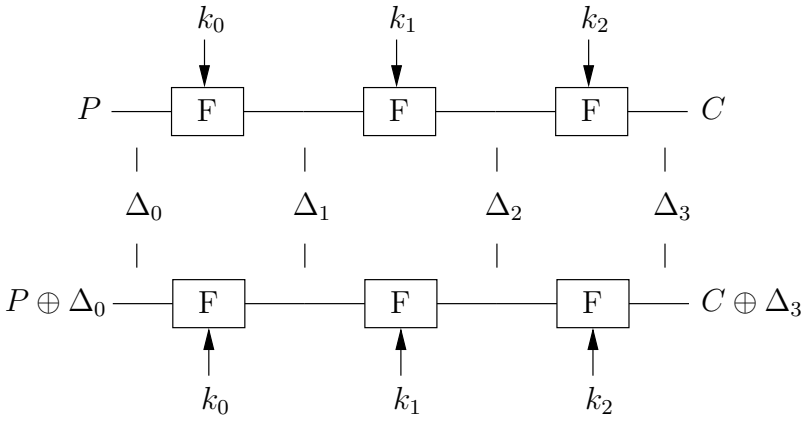

Figure 2.3. A three round differential characteristic starting with difference Δ_0 between the plaintexts, propagating through intermediate differences Δ_1 and Δ_2 to difference Δ_3 between the ciphertexts

Characteristics

As introduced by Biham and Shamir, differential cryptanalysis is based on the concept of differential characteristics.

Definition 2.5. An r-round *differential characteristic* is an $r+1$-tuple $(\Delta_0, \Delta_1, \ldots, \Delta_r)$ specifying the difference between the intermediate values during the encryption of a pair of plaintexts.

See Fig. 2.3 for a representation of a three round differential characteristic.

Starting with a pair of plaintexts, P and $P' = P \oplus \Delta_0$, input to the block cipher using the unknown key k, the attack is based on trying to predict what difference will be created after each successive round. Given a cipher with round function F, the attacker tries to find differences Δ_i and Δ_{i+1} such that the probability of a pair of values with difference Δ_i resulting in a pair of values with difference Δ_{i+1} after application of F is high. That is,

$$\mathrm{Prob}_{P,k}\left\{\mathrm{F}(k,P) \oplus \mathrm{F}(k, P \oplus \Delta_i) = \Delta_{i+1}\right\}$$

should be high, where the probability is taken over all plaintexts and keys.

If (Δ_0, Δ_1) and (Δ_1, Δ_2) are two characteristics each covering one round, a two-round characteristic $(\Delta_0, \Delta_1, \Delta_2)$ can be created by concatenation. Chaining r one-round characteristics an r-round characteristic can be built. A pair of plaintexts that follow the characteristic in each step is called a *right pair* and a pair that does not follow the characteristic is called a *wrong pair*.

The probability of a characteristic can often be approximated by the product of the one-round probabilities. Assuming the subkeys used in the cipher are independent and chosen uniformly at random the probability of a characteristic is equal to

the product of the one-round probabilities. In practice, the subkeys are not independent, making the product of the probabilities an approximation. However, the approximate value is usually close enough to the real value.

By chaining short characteristics together, the attacker might be able to create a characteristic that covers the whole cipher. If the probability of this characteristic is significantly higher than the probability expected for a random permutation the cipher can be distinguished from random.

Using a characteristic that is one, or a few, rounds shorter than the full cipher, an attacker can recover subkeys using the techniques in Section 2.5.2. Additional information on the subkeys used can be gained by analyzing which actual inputs to the round function can cause the observed difference.

If the probability of a characteristic is p, an attack using the characteristic has complexity approximately $\frac{c}{p}$ for some small constant c. A simple way of performing an attack using a characteristic one round shorter than the full cipher is by *counting*, done as follows:

1. Create a counter, initialized to zero, for each of the possible subkey values of the last round.

2. Request the encryption of a pair of plaintexts with the specified input difference.

3. Calculate which subkey values are suggested by this pair of ciphertexts by guessing a subkey value, decrypting the ciphertexts one round and checking if the difference is the one specified by the characteristic. Increment the counter for each of the subkeys that give the right difference.

4. Repeat Step 2 and Step 3 until a few subkeys have significantly higher count than the others.

The expected number of iterations depends on several factors. The probability of the characteristic is one important factor as the attack depends on finding at least one right pair. After $1/p$ trials one right pair is expected. Another factor is the number of subkey values suggested by a wrong pair, as the right value has to be recognized through the noise. For details and improvements, see [9].

Differentials

The basic differential attack uses characteristics, tracing the propagation of differences through the rounds of a cipher. Results may be improved by using differentials [33] instead of differential characteristics.

When a differential characteristic predicts the actual propagation pattern of a difference in the plaintexts to some difference in the ciphertexts, a differential only predicts the difference in the plaintext and in the ciphertext. That is, a differential (Δ_0, Δ_r) can be seen as the collection of all characteristics starting with difference Δ_0 and ending with difference Δ_r. See Fig. 2.4 for a diagram of a three round differential.

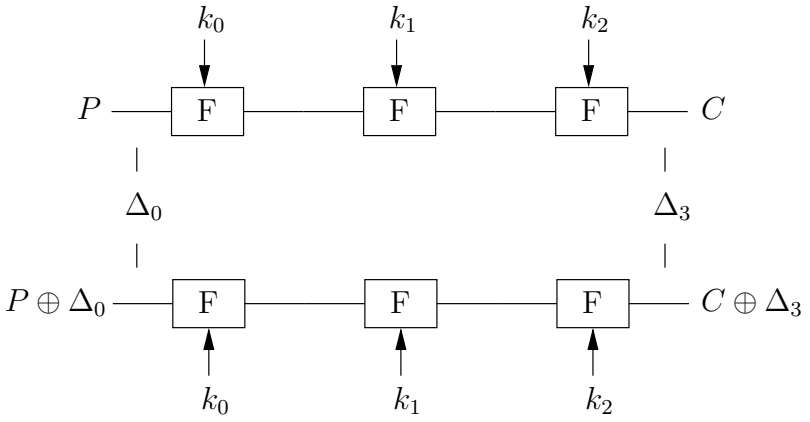

Figure 2.4. A three round differential starting with difference Δ_0 in the plaintexts propagating to difference Δ_3 in the ciphertexts

A differential starting with difference Δ that goes to difference Δ^* is denoted $\Delta \rightarrow \Delta^*$. This notation will also be used for characteristics and truncated differentials as the exact type usually is irrelevant or clear from the context.

Naturally, since a differential contains all possible characteristics with the specified starting and ending difference the probability of a differential is higher than that of the corresponding characteristics. However, there is no known way of finding efficient differentials apart from finding and using a characteristic.

Security Against Differential Cryptanalysis

Since $\frac{c}{p}$ chosen texts are required for a basic differential attack, a cipher can be shown secure against this attack if it can be proved that the best differential has too low probability. Naturally, since there only are 2^N plaintexts for a cipher with block length N, if the probability of the best differential is significantly smaller than 2^{-N} a differential attack is not possible.

For example, for KHAZAD it has been proven [3] that the best 2-round differential characteristic has probability smaller than 2^{-45}. Given the fact that the cipher has 8 rounds and a block size of 64 bits it is highly unlikely that a differential attack using a differential characteristic is possible. However, other attacks, such as boomerang attacks (see Section 2.5.5), using differential characteristics might still be possible.

2.5.4 Truncated Differentials

Truncated differentials were introduced by Knudsen in [29]. The attack is a differential attack and the idea is to predict only part of the difference in the blocks.

Differential cryptanalysis uses predictions of the exact difference in the blocks. However, for some ciphers it is more efficient to fix less information on the differences. Considering the block as a vector with eight components, the attacker might for example specify that the difference should be of the form $(x, -x, 0, y, -y, 0, 2x+y, 0)$

where x and y are any differences with respect to some suitably chosen difference operation. In other cases it might be useful to use wild cards, by for example having a difference of the form $(*, 0, 0, 0, *, *, 0, 0)$ where each $*$ is some unspecified but nonzero difference.

Conventional differential characteristics have the same probability of propagation in both directions, that is, if $(\Delta_0, \Delta_1, \ldots, \Delta_r)$ is an r-round characteristic of probability p in the encryption direction then $(\Delta_r, \ldots, \Delta_1, \Delta_0)$ is an r-round characteristic of probability p in the decryption direction. This is not the case with truncated differentials, which is important for example when using truncated differentials in boomerang attacks.

The wild card nature of truncated differentials allows for the use of pools of plaintexts. If the attacker chooses M plaintexts of the form $(*, c_0, c_1, c_2, *, *, c_3, c_4)$, where the c_i:s are some fixed constants and the $*$ are randomly chosen, the attacker will have $M(M-1)/2$ pairs with difference $(*, 0, 0, 0, *, *, 0, 0)$. This effect is called the *pool effect* and can be used to reduce the number of plaintexts required for an attack. This also means that truncated differentials of probability less than 2^{-N}, where N is the block length, can be used.

Using the pool effect, the filtering of the right pairs from the wrong pairs becomes crucial. A first filtering can usually be done using the predicted output difference and then additional filtering might be done by guessing subkey bytes and checking the difference one round earlier. This is done in Chapter 4 in a boomerang attack using truncated differentials on SAFER++.

2.5.5 Boomerang Attacks

The boomerang attack was introduced by Wagner in [56]. The basic attack uses differential characteristics but does not need them to cover the whole cipher, meaning that ciphers provably secure against differential cryptanalysis might still be breakable using boomerang attacks.

The idea is to find good characteristics for both halfs of a cipher and use them in a quartet structure. See Fig. 2.5 for a picture of a boomerang quartet, similar to the picture in [56]. The attack is a chosen plaintext/adaptively chosen ciphertext attack that can use differentials, differential characteristics or truncated differentials. The basic attack as explained here uses differential characteristics with xor as the difference operation. Chapter 4 contains a boomerang attack on a reduced version of SAFER++ using a combination of differential characteristics and truncated differentials.

Let the cipher E be the composition of functions E_0 and E_1, that is $\mathrm{E} = \mathrm{E}_1 \circ \mathrm{E}_0$. Assume that $\Delta \to \Delta^*$ is a characteristic of probability p for E_0 and that $\nabla \to \nabla^*$ is a characteristic of probability q for E_1^{-1}. The idea is to find a pair of plaintexts P, $P' = P \oplus \Delta$, such that the characteristic $\Delta \to \Delta^*$ holds for the first half of the encryption. That is, $\mathrm{E}_0(P) \oplus \mathrm{E}_0(P') = \Delta^*$, which happens with probability p. Let the encryptions of P and P' be $C = \mathrm{E}(P)$ and $C' = \mathrm{E}(P')$. Create two new ciphertexts $D = C \oplus \nabla$ and $D' = C' \oplus \nabla$. With probability q^2 the characteristic

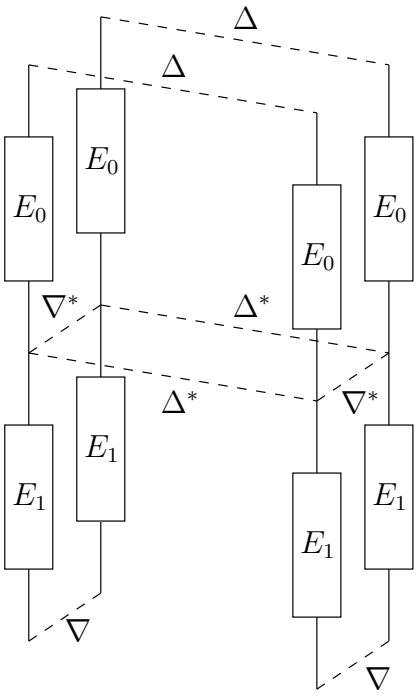

Figure 2.5. The boomerang quartet

$\nabla \to \nabla^*$ holds in the first half of the decryption of the two pairs C, D and C', D'. That is, $\mathrm{E}_1^{-1}(C) \oplus \mathrm{E}_1^{-1}(D) = \mathrm{E}_0(P) \oplus \mathrm{E}_1^{-1}(D) = \nabla^*$ and $\mathrm{E}_1^{-1}(C') \oplus \mathrm{E}_1^{-1}(D') = \mathrm{E}_0(P') \oplus \mathrm{E}_1^{-1}(D') = \nabla^*$. Now the following holds:

$$\mathrm{E}_1^{-1}(D) \oplus \mathrm{E}_1^{-1}(D') = \mathrm{E}_1^{-1}(D) \oplus \mathrm{E}_0(P) \oplus \mathrm{E}_0(P) \oplus \mathrm{E}_0(P') \oplus$$
$$\oplus \mathrm{E}_0(P') \oplus \mathrm{E}_1^{-1}(D') = \nabla^* \oplus \Delta^* \oplus \nabla^* = \Delta^*$$

which shows that the difference between $\mathrm{E}_1^{-1}(D)$ and $\mathrm{E}_1^{-1}(D')$ is set up to use the characteristic $\Delta \to \Delta^*$ backwards, that is $\Delta^* \to \Delta$ with probability p. The result is that the difference Δ appears between the decryptions Q and Q' of D and D'.

In total, the probability of the boomerang is p^2q^2, which might be significantly higher than that of the best characteristic covering the whole cipher.

Since there are two pairs in a boomerang quartet, the filtering power of the attacker is essentially doubled. Guessing some relevant bytes of the subkey at the top, the attacker has two pairs to use for filtering instead of one as in a differential attack.

The advantages of the boomerang attack come at the expense of needing both chosen plaintext and adaptively chosen ciphertext. Improvements of the boomerang attack, called amplified boomerang and rectangle attacks respectively are in [25] and [7].

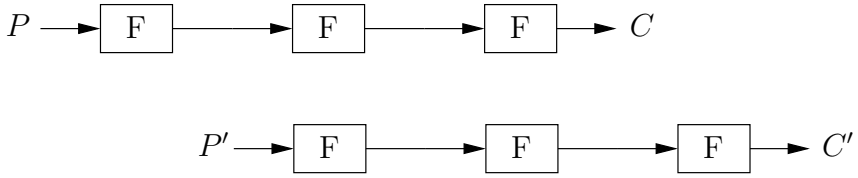

Figure 2.6. A properly slid pair in the slide attack

2.5.6 Slide Attacks and Slide with a Twist

Slide attacks were introduced by Biryukov and Wagner in [14]. They are applicable to iterated block ciphers in which exactly the same round function is reused with some period.

Assume that the block cipher E can be written as some number r of iterations of a function F, that is $\mathrm{E} = \mathrm{F} \circ \mathrm{F} \circ \ldots \circ \mathrm{F} = \mathrm{F}^{\mathrm{r}}$. The iterated function F does not depend on any subkeys in this scenario. That is, every application of F gives the same intermediate block out, given the same intermediate block in. This can happen for example if the key schedule is periodic, that is, generates a repeating sequence of subkeys.

The attack tries to find a pair of plaintexts, P, P' such that the encryptions of P and P' are offset by one application of F (see Fig. 2.6). That is, $\mathrm{F}(P) = P'$. Let $C = \mathrm{E}(P)$ and $C' = \mathrm{E}(P')$. Now, since $P' = \mathrm{F}(P)$ and $\mathrm{E} = \mathrm{F}^{\mathrm{r}}$ we have that $C' = \mathrm{E}(P') = \mathrm{E}(\mathrm{F}(P)) = \mathrm{F}^{\mathrm{r}}(\mathrm{F}(P)) = \mathrm{F}(\mathrm{F}^{\mathrm{r}}(P)) = \mathrm{F}(C)$

Since the round function of a block cipher usually is quite weak with respect to known pairs of input and output values the cipher can be broken. Given the plaintexts P, P' and the ciphertexts C, C' the attacker has two pairs of values to use to break the round function and recover the key.

The attack depends on finding collisions in the intermediate values of the cipher. To create the interesting pairs the attacker uses the birthday paradox. Given $2^{N/2}$ random known plaintexts, there are approximately 2^N candidate pairs. Each pair P, P' has probability 2^{-N} to satisfy $P' = \mathrm{F}(P)$, where N is the block length, so the attacker expects one such pair. This calculation is only an approximation since the pairs are not independent, but it works well enough in practice.

The detection of the properly *slid pair* from the noise is done in different ways depending on the cipher under attack. For a Feistel cipher with one round self-similarity, which uses the same function in each round, candidate slid pairs can be recognized using the fact that one half of each block is equal in a slid pair. For other ciphers it might be possible to try and solve for the unknown key and to reject a pair that gives a contradiction.

Slide with a Twist

In [15] a variant of the slide attack, called slide with a twist was introduced. The attack was described against a Feistel cipher with two round self-similarity, that is, the key is reused every other round. Due to the symmetry of encryption and

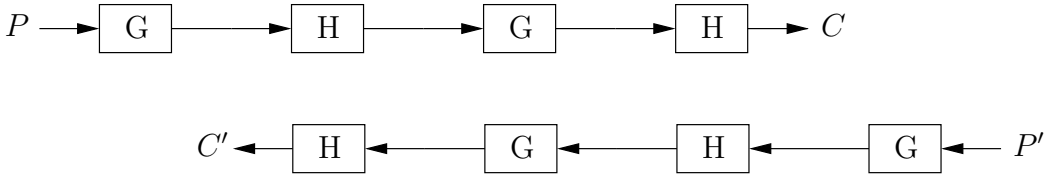

Figure 2.7. A properly slid pair in the slide with a twist. Note that the functions G and H are involutions

decryption in a Feistel cipher this is similar to writing the encryption function E as the iterated composition of two involutions G and H, $\mathrm{E} = (\mathrm{HG})^r$. Compared to the regular slide attack the slide with a twist slides an instance of encryption against an instance of decryption instead of encryption against encryption.

In detail, the slide with a twist is based on finding two plaintext/ciphertext pairs P, C and P', C' such that $C = \mathrm{E}(P)$, $C' = \mathrm{E}(P')$ and $C' = \mathrm{G}(P)$. Then it also holds that $C = (\mathrm{HG})^r(P) = (\mathrm{HG})^{r-1}(\mathrm{H}(\mathrm{G}(P))) = (\mathrm{HG})^{r-1}(\mathrm{H}(C'))$. Rewriting to $C = \mathrm{H}((\mathrm{GH})^{r-1}(C'))$ and applying G the result is $\mathrm{G}(C) = (\mathrm{GH})^r(C') = P'$. See Fig. 2.7 for a pictorial representation of a properly slid pair in the slide with a twist.

2.5.7 Linear Cryptanalysis

Linear cryptanalysis is a known plaintext attack introduced by Matsui in [39]. The attack is based on building linear approximations to the round function of a block cipher and using them to recover information on the key.

Let $A[i]$ be the i:th bit of A and $A[a_0, a_1, \ldots, a_n]$ be the xor of bits a_0, a_1, $\ldots$, a_n of A. A linear approximation of the round function F using key K with $Y = \mathrm{F}(K, X)$ is an expression of the form $X[c_1, c_2, \ldots, c_l] \oplus Y[d_1, d_2, \ldots, d_m] = K[e_1, e_2, \ldots, e_n]$ which holds with probability $1/2 + b$, for some b, $-1/2 \leq b \leq 1/2$. The bias b measures the effectiveness of the approximation. Such an approximation is denoted by $(\Delta_X, \Delta_Y, \Lambda_K)$ where $\Delta_X = [c_1, c_2, \ldots, c_l]$, $\Delta_Y = [d_1, d_2, \ldots, d_m]$ and $\Lambda_K = [e_1, e_2, \ldots, e_n]$.

By chaining approximations together, an approximation for the whole cipher can be created. Two approximations, $(\Delta_X, \Delta_Y, \Lambda_K)$ and $(\Delta_{X'}, \Delta_{Y'}, \Lambda_{K'})$, can be used to approximate $\mathrm{F}(K_2, \mathrm{F}(K_1, X))$ if $\Delta_Y = \Delta_{X'}$, that is, if the bits selected from the output of the function in the first approximation are the same as the ones used at the input of the function in the second approximation. See Fig. 2.8 for an example of the chaining of approximations.

If the first approximation $(\Delta_0, \Delta_1, \Lambda_K)$ has bias b_1 and the second approximation $(\Delta_1, \Delta_2, \Lambda_{K'})$ has bias b_2, by applying the piling-up lemma [39] the bias of the chained approximation $(\Delta_0, \Delta_2, \Lambda_K \oplus \Lambda_{K'})$ can be approximated by $2b_1b_2$.

The basic attack on a cipher using the linear approximation $(\Delta_P, \Delta_C, \Lambda_K)$ with bias b is done by collecting a large amount of plaintexts and the corresponding ciphertexts, calculating the value of the approximation and then using the value of $K[\Lambda_K]$ that is most likely. The number of texts needed is on the order of b^{-2}.

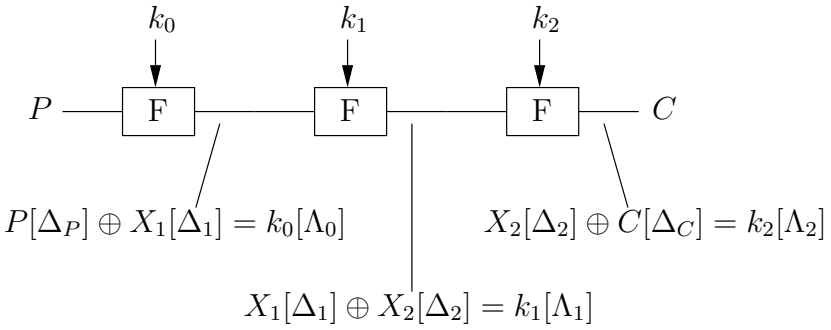

Figure 2.8. Chaining of three approximations results in the approximation $P[\Delta_P] \oplus C[\Delta_C] = k_0[\Lambda_0] \oplus k_1[\Lambda_1] \oplus k_2[\Lambda_2]$ for the three round cipher. Note that the intermediate values $X_i[\Delta_i]$ are all canceled out in the xor of the approximations

Linear cryptanalysis can also be used in attacks based on the ideas in Section 2.5.2. Such an attack, that was the first attack carried out in practice against DES, is in [40].

2.5.8 Multiset Attacks

The multiset attack, also known as the SQUARE attack and the saturation attack, was first applied to the cipher SQUARE [18]. In this context a multiset is a set where a value is allowed to appear several times. The multiplicity of each value is one important characteristic of a multiset. We consider multisets where the elements are m bit values, for some suitable value of m.

We use the notation of [13] and restate the following four properties that can be used to characterize a multiset:

Constant multisets (denoted by C) are multisets where all elements are equal.

Permutation multisets (denoted by P) are multisets in which every possible value appears exactly once.

Even multisets (denoted by E) are multisets where each value in the multiset appears an even number of times.

Balanced multisets (denoted by B) are multisets where the sum of the values in the multiset is zero. The summation operator used can vary depending on the cipher being studied and is often xor or modular addition.

The multiset attack tries to trace the evolution of some specific multiset through the rounds of a cipher and use it to recover information on the key using the following lemma:

Lemma 2.6 [13]. The following properties hold:

1. A multiset with either property P or E also has property B when $m > 1$.

2. Properties E and C are preserved by all functions on m bit values.

3. Property P is preserved by all bijections on m bit values.

4. Property B is preserved by all affine mappings from m bits to n bits when $m > 1$ and the size of the multiset is even.

Now we consider sets of blocks of qm bits where each individual m bit component can be viewed as a multiset. For example, we denote by $(\mathrm{P}, \mathrm{C}, \mathrm{C}, \ldots, \mathrm{C})$ a set of blocks in which the first m bit component takes every possible value and the other $q-1$ m bit components are equal in each block.

Consider a block cipher where the block under encryption is a vector in $\mathrm{GF}(2^m)^q$. Assume that the round function of the block cipher consists of the parallel application of a bijective $m \times m$ S-box to each component of the vector followed by a linear $\mathrm{GF}(2^m)^q \rightarrow \mathrm{GF}(2^m)^q$ transformation with branch $q+1$ and a key application using vector addition in $\mathrm{GF}(2^m)^q$. The balancedness property will be with respect to addition in $\mathrm{GF}(2^m)$.

In a chosen plaintext multiset attack the attacker now chooses 2^m plaintexts of the form $(\mathrm{P}, \mathrm{C}, \mathrm{C}, \ldots, \mathrm{C})$, that is a set of plaintexts in which the first m bit component of the plaintexts takes on every possible value and the other components are constant. The first nonlinear layer preserves the multiset which is then transformed into $(\mathrm{P}, \mathrm{P}, \ldots, \mathrm{P})$ by the linear transformation since each component of the output from the linear transform can be written as a linear combination of the input components of which all but one are constant. The key addition and the second nonlinear layer preserves the multiset. The second linear transformation results in a multiset of the form $(\mathrm{B}, \mathrm{B}, \ldots, \mathrm{B})$ since the input is balanced in each component and each output component is a linear combination of the input components. The balancedness property is preserved by the key addition, but generally destroyed by the next nonlinear layer. However, attacks using the ideas of Section 2.5.2 can be performed. A correct guess of the keys applied after the nonlinear layer that destroyed the balancedness property can be identified since the set of values found by partial decryption should be balanced.

The multiset attack and variations have been applied to several ciphers. Some results are in [13], [22] and [34].

2.5.9 Weak Keys

Weak keys are keys whose use make an attack on the cipher easier or gives the cipher some other undesirable property. DES, for example, has four weak keys that have the property that encryption is the same as decryption and eight pairs of keys that can decrypt plaintexts encrypted with the other key in the pair. IDEA have several large classes of keys that make the cipher weak under different attacks [17][23][11].

Weak keys are undesirable for several reasons. If the class of weak keys is large, a randomly selected key might be weak allowing an attack. Even a small class of weak keys might have implications since they could be exploited for example when

the cipher is used to build a hash function. See [28] for an example of a weak key schedule leading to weaknesses in the cipher when used in a hashing mode.

The cardinality of the set of weak keys is important since it imposes a limit on the time complexity of an attack. If the set of weak keys contains M keys, any attack exploiting the weak keys must have time complexity significantly lower than time equivalent to M encryptions as that is the complexity of exhaustive key search.

Apart from recovering the key, attacks in the context of weak keys can be *membership tests*. Membership tests are attacks designed not to recover the unknown key, but to determine if the key is a member of the set of weak keys.

2.6 Chapter Summary

In this chapter the basic theory of block ciphers has been covered. Block ciphers were defined as functions from a set of plaintexts and a set of keys to a set of ciphertexts. Some common ways of constructing block ciphers, such as substitution permutation networks and the Feistel structure, were presented together with information on the building blocks of round functions.

Classifications of attacks on block ciphers according to the result of the attack and the requirements of the attacker were given followed by some discussion on the appropriate measures of attack complexity. The chapter ended with an introduction to several cryptanalytic attacks that are applicable to block ciphers. Among the attacks covered were slide attacks and boomerang attacks that will be used in our analysis of KHAZAD and SAFER++ respectively.

Chapter 3

Analysis of Khazad

This chapter contains our analysis of the block cipher KHAZAD. After a brief introduction a description of the cipher and the key schedule is given. As the nonlinear layer of KHAZAD is highly nonlinear and the linear layer has a high rate of diffusion classical attacks such as differential cryptanalysis are hard to mount. Instead we focus on the properties resulting from the involutional structure. The main result of the chapter is the existence of 2^{64} weak keys in 5 round KHAZAD resulting from the involutional structure.

3.1 The Cipher Khazad

The block cipher KHAZAD [3] was invented by Barreto and Rijmen and submitted to the NESSIE project [50] in 2000. It has a 64-bit block, a 128-bit key and 8 rounds.

KHAZAD is a substitution linear network with an involutional structure. By choosing all components of the cipher to be involutions the encryption and decryption structure of the cipher differ by a change in the key schedule only. The involutional structure of the cipher gives it several interesting mathematical properties. The block cipher ANUBIS [2] is closely related to KHAZAD as it also has an involutional structure and is designed according to the same principles as KHAZAD.

Using the known branch number of the linear transformation and the probability of the best differential through the nonlinear layer the designers show that the best differential characteristic for two rounds of KHAZAD has probability not larger than 2^{-45}. Similarly, the bias of the best linear approximation over two rounds is not larger than 2^{-18}. The designers state that "This makes classical differential or linear attacks, as well as some advanced variants like differential-linear attacks, very unlikely to succeed for the full cipher." [3]. Furthermore, the designers claim resistance against truncated differential attacks, interpolation attacks and related key cryptanalysis and claim that there are no weak keys.

The round function and the key schedule are based on three involutions: the nonlinear layer, the linear layer and the key addition. The block under encryption is represented as a vector in $\mathrm{GF}(2^8)^8$ with the finite field $\mathrm{GF}(2^8)$ represented as

$\mathrm{GF}(2)[x]/p_8(x)$ where $p_8(x) = x^8 + x^4 + x^3 + x^2 + 1$. We write the elements of $\mathrm{GF}(2^8)$ using hexadecimal notation. For example, the element $x^7 + x^5 + 1$ is written $\mathtt{a1}_x$.

3.1.1 The Nonlinear Layer

The nonlinear layer of KHAZAD applies a nonlinear substitution to the components of the block. A function $\mathrm{s} : \mathrm{GF}(2^8) \to \mathrm{GF}(2^8)$ is applied in parallel resulting in a function $\mathrm{S} : \mathrm{GF}(2^8)^8 \to \mathrm{GF}(2^8)^8$.

The function s is an involution built from two $\mathrm{GF}(2^4) \to \mathrm{GF}(2^4)$ involutions P and Q. The functions P, Q and s are given in Appendix A. The nonlinear layer of KHAZAD will be denoted by S. Since s is an involution and S consists of the parallel application of s, the function S is an involution.

3.1.2 The Linear Layer

The linear layer, denoted by M, is a $\mathrm{GF}(2^8)^8 \to \mathrm{GF}(2^8)^8$ operation based on matrix multiplication in $\mathrm{GF}(2^8)$. Given the matrix

$$\boldsymbol{M} = \begin{bmatrix} \mathtt{01}_x & \mathtt{03}_x & \mathtt{04}_x & \mathtt{05}_x & \mathtt{06}_x & \mathtt{08}_x & \mathtt{0B}_x & \mathtt{07}_x \\ \mathtt{03}_x & \mathtt{01}_x & \mathtt{05}_x & \mathtt{04}_x & \mathtt{08}_x & \mathtt{06}_x & \mathtt{07}_x & \mathtt{0B}_x \\ \mathtt{04}_x & \mathtt{05}_x & \mathtt{01}_x & \mathtt{03}_x & \mathtt{0B}_x & \mathtt{07}_x & \mathtt{06}_x & \mathtt{08}_x \\ \mathtt{05}_x & \mathtt{04}_x & \mathtt{03}_x & \mathtt{01}_x & \mathtt{07}_x & \mathtt{0B}_x & \mathtt{08}_x & \mathtt{06}_x \\ \mathtt{06}_x & \mathtt{08}_x & \mathtt{0B}_x & \mathtt{07}_x & \mathtt{01}_x & \mathtt{03}_x & \mathtt{04}_x & \mathtt{05}_x \\ \mathtt{08}_x & \mathtt{06}_x & \mathtt{07}_x & \mathtt{0B}_x & \mathtt{03}_x & \mathtt{01}_x & \mathtt{05}_x & \mathtt{04}_x \\ \mathtt{0B}_x & \mathtt{07}_x & \mathtt{06}_x & \mathtt{08}_x & \mathtt{04}_x & \mathtt{05}_x & \mathtt{01}_x & \mathtt{03}_x \\ \mathtt{07}_x & \mathtt{0B}_x & \mathtt{08}_x & \mathtt{06}_x & \mathtt{05}_x & \mathtt{04}_x & \mathtt{03}_x & \mathtt{01}_x \end{bmatrix}$$

and a vector $\boldsymbol{a} \in \mathrm{GF}(2^8)^8$ the result of the linear layer is $\mathrm{M}(\boldsymbol{a}) = \boldsymbol{a}\boldsymbol{M}$. The function M is an involution.

3.1.3 The Key Addition

The round key addition of KHAZAD is done using xor. Given the block $\boldsymbol{a} \in \mathrm{GF}(2^8)^8$ and a subkey $\boldsymbol{k}_i \in \mathrm{GF}(2^8)^8$ the key addition is $\mathrm{K}[\boldsymbol{k}_i](\boldsymbol{a}) = \boldsymbol{k}_i \oplus \boldsymbol{a}$. We denote the application of subkey $\boldsymbol{k}_i$ by k_i, that is, $\mathrm{k}_i = \mathrm{K}[\boldsymbol{k}_i]$. Since xor is an involution, the function k_i is an involution.

3.1.4 The Round Function

The round function of KHAZAD is built from the functions described previously. The round function $\rho[\boldsymbol{k}_i]$ applies the nonlinear function S followed by the linear transformation M and the subkey addition k_i. That is, $\rho[\boldsymbol{k}_i] = \mathrm{K}[\boldsymbol{k}_i] \circ \mathrm{M} \circ \mathrm{S} = \mathrm{k}_i\mathrm{MS}$.

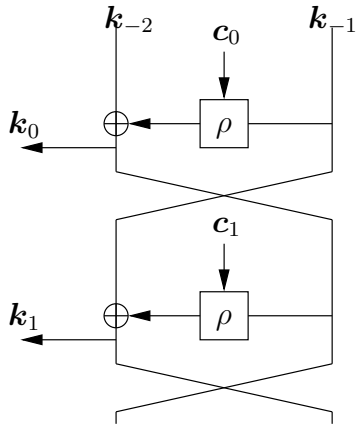

Figure 3.1. Generation of the first two subkeys k_0 and k_1 from the master key k_{-2} and k_{-1} in the key schedule of KHAZAD using the round function ρ and the constants c_0 and c_1

3.1.5 The Key Schedule

The key schedule expands the 128-bit key, represented as a vector in $\mathrm{GF}(2^8)^{16}$, to the required number of subkeys.

The key is split into two halves, k_{-2} and k_{-1}, and the subkeys are generated using the round function ρ in a Feistel cipher structure with a set of constants as the subkeys (see Fig. 3.1). Given a set of constants, c_i derived from the S-box, the subkey k_i is $k_i = \rho[c_i](k_{i-1}) \oplus k_{i-2}$. Since KHAZAD has 8 rounds and one subkey is used before the first round the key schedule is used to generate 9 subkeys.

3.1.6 The Full Cipher

Given the subkeys $k_0, k_1, \ldots, k_8$ the cipher KHAZAD is defined as $\mathrm{k}_8 \circ \mathrm{S} \circ \rho[k_7] \circ \rho[k_6] \circ \ldots \circ \rho[k_1] \circ \mathrm{k}_0$. That is, k_0 is applied before the first round and the last round leaves the linear layer M out.

3.2 Properties

This section describes some properties observed in KHAZAD. The properties result from the linearity of the key addition and the linear layer in connection with the involutional structure.

3.2.1 Diffusion in the Linear Layer

The linear layer of KHAZAD is based on an error-correcting MDS code. The branch number of the $\mathrm{GF}(2^8)^8$ linear transformation M is 9, as shown in [3]. That is, the sum of changed components of the input vector and the output vector of the linear transformation is at least 9.

The known branch number allows the designers to calculate a bound on the maximum probability of a two round differential characteristic and the maximum bias of a two round linear approximation. As noted in the introduction, it is unlikely that classical differential och linear cryptanalysis works for the full KHAZAD cipher.

3.2.2 Linearity

As the linear layer and the key addition are both involutions and linear with respect to xor the following holds:

Lemma 3.1. By linearity, $\mathrm{k}_i \circ \mathrm{M} = \mathrm{M} \circ \mathrm{k}'_i$ where $\mathrm{k}'_i(\boldsymbol{x}) = \boldsymbol{k}_i\boldsymbol{M} \oplus \boldsymbol{x}$.

Proof. It is sufficient to note that $(\mathrm{k}_i \circ \mathrm{M})(\boldsymbol{x}) = \boldsymbol{k}_i \oplus \boldsymbol{x}\boldsymbol{M} = (\boldsymbol{k}_i\boldsymbol{M} \oplus \boldsymbol{x})\boldsymbol{M} = (\mathrm{M} \circ \mathrm{k}'_i)(\boldsymbol{x})$ since M is an involution. ∎

This shows that the key addition can be moved through the linear layer M due to linearity. This is the reason for the last round to leave M out. We will use it to find an equivalent representation of KHAZAD more suitable for application of the slide with a twist technique.

3.2.3 Involutional Structure

The involutional structure of KHAZAD together with the linear property of the previous section allows us to rewrite KHAZAD in several interesting ways.

For example, 5 round KHAZAD, which is the main target of our analysis, can be rewritten as

$$\mathrm{k}_5\mathrm{Sk}_4\mathrm{MSk}_3\mathrm{MSk}_2\mathrm{MSk}_1\mathrm{MSk}_0 = \mathrm{k}_5\mathrm{Sk}_4\mathrm{MSMk}'_3\mathrm{Sk}_2\mathrm{MSMk}'_1\mathrm{Sk}_0$$

or for clarity

$$\mathrm{k}_5\mathrm{Sk}_4\,[\mathrm{MSM}]\,\mathrm{k}'_3\mathrm{Sk}_2\,[\mathrm{MSM}]\,\mathrm{k}'_1\mathrm{Sk}_0 \tag{3.1}$$

It can be noted that MSM is an involution since both M and S are involutions.

3.3 Previous Attacks

3.3.1 Multiset Attack on 3 and 4 Rounds

Up to 4 rounds of KHAZAD can be broken with a multiset attack as shown in [3]. The attack is a standard multiset attack using a total of 2^9 chosen plaintexts.

The attack on 3 rounds is as follows. A set of 2^8 plaintexts is created where the plaintexts are all different in one byte and constant in the other bytes. That is, one byte takes all 2^8 different values and the other bytes are constant. In the notation of Section 2.5.8 we have plaintexts of the form $(\mathrm{P}, \mathrm{C}, \ldots, \mathrm{C})$. After the first key addition and the first nonlinear layer this is still $(\mathrm{P}, \mathrm{C}, \ldots, \mathrm{C})$ which is turned into $(\mathrm{P}, \mathrm{P}, \ldots, \mathrm{P})$ by the linear layer of the first round. The next round turns this into

balanced sets $(\mathrm{B}, \mathrm{B}, \ldots, \mathrm{B})$ and in the last round the balancedness is destroyed by the nonlinear layer. Now, the key recovery attack of Section 2.5.2 can be performed.

Guessing a key byte of the last round, partially decrypting through the last round and checking the balancedness property can be used to find one key byte. With high probability two sets of 2^8 texts is enough to distinguish a correct key guess from an incorrect, since each set gives an 8-bit condition. The attack can then be performed in turn for the other key bytes of the last round.

To recover one key byte the attacker needs to perform 2^8 key guesses and for each key guess check a condition in a set of 2^8 texts, giving a total complexity of 2^{16} S-box lookups. The few remaining wrong keys can be filtered out using the second set of 2^8 plaintexts.

Extending the attack to 4 rounds is claimed to be possible by guessing the full 64-bit key of the last round, decrypting one round and then performing the 3 round attack as above. The complexity is thus $2^{64} \times 2^{16} = 2^{80}$ operations.

3.3.2 Permutation Cycle Structure

In [12] the structure of KHAZAD is investigated. Using the involutional structure a number of properties are described. We restate some of the analysis here.

The analysis relies on two theorems regarding the cycle structure of the composition of involutions. The first is due to Rejewski [52] and was central to the cryptanalysis of the Enigma machine during World War II.

Theorem 3.2 [52]. The number of cycles of each length in the composition of two involutions without fixed points is an even number.

Theorem 3.3 [12]. For any involution I, $\mathrm{k}_j\mathrm{Ik}_i$ can be written as the composition of two involutions.

Proof. Let $\Delta = \mathrm{k}_j\mathrm{k}_i$. Δ is an involution since xor is an involution. It is sufficient to note that $\mathrm{k}_j\mathrm{Ik}_i = \mathrm{k}_j\mathrm{Ik}_j \circ \Delta$ and since k_j is an involution, $\mathrm{k}_j\mathrm{Ik}_j$ is an involution, thus completing the proof. ∎

As noted previously (Equation 3.1) 5 round KHAZAD can be written as

$$\mathrm{k}_5\mathrm{Sk}_4\,[\mathrm{MSM}]\,\mathrm{k}'_3\mathrm{Sk}_2\,[\mathrm{MSM}]\,\mathrm{k}'_1\mathrm{Sk}_0$$

Using Theorem 3.3 it is shown that $\mathrm{k}'_3\mathrm{Sk}_2$ is a permutation whose cycle structure, that is the number and lengths of cycles, depends only on $\boldsymbol{k}_2 \oplus \boldsymbol{k}'_3$. Furthermore, each cycle length appears an even number of times, the longest cycle is shorter than 2^{56} and the cycle lengths are 128-smooth numbers, meaning that the largest prime factor of the cycle length is not larger than 128.

For randomly chosen keys $\boldsymbol{k}_2$ and $\boldsymbol{k}'_3$ the function $\mathrm{k}'_3\mathrm{Sk}_2$ has no fixed points with probability approximately $2^{-10.6}$. If there are fixed points at all, there are at least 2^8 of them.

Since MSM is an involution the composition $[\mathrm{MSM}]\,\mathrm{k}'_3\mathrm{Sk}_2\,[\mathrm{MSM}]$ is conjugate to $\mathrm{k}'_3\mathrm{Sk}_2$ and therefore has the same cycle structure. This covers 3.5 rounds of KHAZAD.

An interesting property in 5 round KHAZAD also described in [12] is the following. Denoting 5 round KHAZAD by E_5 and 3 round KHAZAD by E_3 the following holds. If an attacker guesses $\boldsymbol{\Delta}_k = \boldsymbol{k}_0 \oplus \boldsymbol{k}_5$ and iterates $(\mathrm{E}_5(\boldsymbol{x}) \oplus \boldsymbol{\Delta}_k)$ the result is almost equivalent to iteration of E_3 due to cancellations. That is,

$$(\mathrm{E}_5(\boldsymbol{x}) \oplus \boldsymbol{\Delta}_k)^n = \mathrm{k}_0\mathrm{SM} \circ (\mathrm{E}_3)^n \circ \mathrm{MSk}_0(\boldsymbol{x})$$

If the cycle structure of 3 round KHAZAD can be distinguished from random in less than 2^{64} steps, this property might allow 5 round KHAZAD to be broken. We have not found a way of doing this and are not aware of any other such results.

3.4 Weak Keys Under Slide with a Twist

In this section we apply the slide with a twist technique, described in Section 2.5.6, to 5 rounds of KHAZAD under a set of weak keys. We show that the key can be recovered faster than exhaustive search and consider extensions to more rounds and more keys.

3.4.1 Sliding 5 Rounds

As shown previously, 5 rounds of KHAZAD can be written as

$$\mathrm{k}_5\mathrm{Sk}_4\,[\mathrm{MSM}]\,\mathrm{k}'_3\mathrm{Sk}_2\,[\mathrm{MSM}]\,\mathrm{k}'_1\mathrm{Sk}_0$$

Since [MSM] is an involution, 5 rounds of KHAZAD has a clear symmetry with respect to encryption and decryption. We write the encryption of plaintext P_0 into ciphertext C_0 and the decryption of ciphertext C_1 into plaintext P_1 as

$$C_0 \leftarrow \mathrm{k}_5\mathrm{Sk}_4 \qquad [\mathrm{MSM}]\mathrm{k}'_3\mathrm{Sk}_2\,[\mathrm{MSM}] \qquad \mathrm{k}'_1\mathrm{Sk}_0 \leftarrow P_0 \tag{3.2}$$

$$P_1 \leftarrow \mathrm{k}_0\mathrm{Sk}'_1 \qquad [\mathrm{MSM}]\mathrm{k}_2\mathrm{Sk}'_3\,[\mathrm{MSM}] \qquad \mathrm{k}_4\mathrm{Sk}_5 \leftarrow C_1 \tag{3.3}$$

noting the structural symmetry. For example, the middle part, $[\mathrm{MSM}]\,\mathrm{k}_2\mathrm{Sk}'_3\,[\mathrm{MSM}]$ is highly symmetrical. This symmetry of encryption and decryption is the property needed for application of the slide with a twist.

Restricting the key used so that $\boldsymbol{k}_2 = \boldsymbol{k}'_3$ makes $[\mathrm{MSM}]\,\mathrm{k}_2\mathrm{Sk}'_3\,[\mathrm{MSM}]$ an involution. There are 2^{64} keys with this property and we will show that these keys are weak with respect to a slide with a twist attack.

3.4.2 Equations for 5 Rounds

From now on, we assume that the subkeys $\boldsymbol{k}_2$ and $\boldsymbol{k}'_3$ are equal, that is, $\boldsymbol{k}_2 = \boldsymbol{k}'_3$. Note that this makes $[\mathrm{MSM}]\,\mathrm{k}_2\mathrm{Sk}'_3\,[\mathrm{MSM}]$ an involution without fixed points since it is conjugate to S, which is an involution without fixed points.

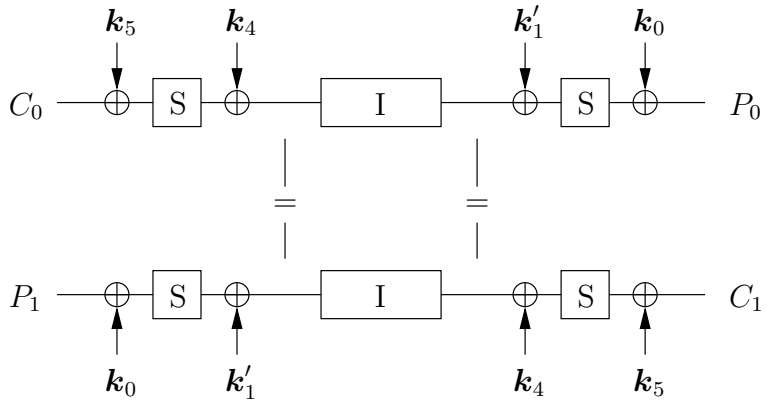

Figure 3.2. A proper pair in the slide with a twist on 5 round KHAZAD with a weak key. S is the S-box layer, $\boldsymbol{k}_i$ are the subkeys and I is an involution resulting from the weak key. Equal values to the left of the involution I results in equal values to the right of the involution and vice versa

Using Equations 3.2 and 3.3 we see that if $\mathrm{k}'_1\mathrm{Sk}_0(P_0) = \mathrm{k}_4\mathrm{Sk}_5(C_1)$ for some plaintext P_0 and some ciphertext C_1, the values before application of the involution $[\mathrm{MSM}]\,\mathrm{k}_2\mathrm{Sk}'_3\,[\mathrm{MSM}]$ are equal. Being an involution, the values must be equal after the application too. That is, $\mathrm{k}_4\mathrm{Sk}_5(C_0) = \mathrm{k}'_1\mathrm{Sk}_0(P_1)$. Such a pair of plaintexts and ciphertexts is a right pair in our slide with a twist attack. In Fig. 3.2 a pictorial representation of a right pair is given.

In summary, for a proper pair we have

$$\mathrm{k}'_1\mathrm{Sk}_0(P_0) = \mathrm{k}_4\mathrm{Sk}_5(C_1) \tag{3.4}$$

$$\mathrm{k}'_1\mathrm{Sk}_0(P_1) = \mathrm{k}_4\mathrm{Sk}_5(C_0) \tag{3.5}$$

These equations are central to the attack. We note that S consists of the parallel application of a $\mathrm{GF}(2^8) \to \mathrm{GF}(2^8)$ function with no interaction between the components. Furthermore, the actual values of $\boldsymbol{k}'_1$ and $\boldsymbol{k}_4$ cannot be recovered, only their xor.

3.4.3 Generating Slid Pairs

Potential slid pairs can be generated using the birthday paradox. In a set of 2^{32} random plaintexts there are approximately 2^{63} pairs each having probability 2^{-64} of being a slid pair. Since the set of weak keys considered consists of 2^{64} keys, an attack must have complexity lower than 2^{64} as that is the complexity of exhaustive search. Clearly, this makes considering all 2^{63} pairs generated by a set of 2^{32} plaintexts hard. We use a different approach, using the observation above that S does not mix the components of the block.

From Equations 3.4 and 3.5 we get that if P_0 and P_1 are equal in some components, C_0 and C_1 will be equal in the same components if the pair is a slid pair. For example, if the xor of P_0 and P_1 is of the form $(0,0,0,0,*,*,*,*)$ the xor of C_0 and C_1 will also be of the form $(0,0,0,0,*,*,*,*)$.

A set of 2^{32} plaintexts of the form $(0,0,0,0,*,*,*,*)$, where the first 32 bits are constant and the last 32 bits take every possible value will generate 2^{63} pairs and also give a reasonably efficient filter for incorrect pairs. For a right pair P_0 and P_1 of plaintexts, the ciphertexts C_0 and C_1 must have a difference of the form $(0,0,0,0,*,*,*,*)$. Since the block size of KHAZAD is 64 bits, this gives a 32 bit filter for incorrect pairs. However, since there are 2^{63} pairs a 32 bit filter still leaves 2^{31} pairs.

Since there are two 64-bit equations and only 64 bits of freedom for the key, there is enough information to solve for the key, but the complex relation between the subkeys makes actually solving the system hard. Generating more slid pairs is one solution to this problem.

3.4.4 Probability of Slid Pairs

In a pool of the form introduced in the previous section, the probability of a pair being a slid pair can be calculated as follows. Let $X_i = [\mathrm{MSM}]\,\mathrm{k}'_3\mathrm{Sk}_2\,[\mathrm{MSM}]\,\mathrm{k}'_1\mathrm{Sk}_0\,(P_i)$ and $Y_j = \mathrm{k}'_1\mathrm{Sk}_0\,(P_j)$. The required condition for the pair to be a slid pair is that $X_i = Y_j$, which can be seen by considering Equations 3.2 and 3.3.

It can be noted that the Y_j:s created by all plaintexts P_j are of the form $(c_0, c_1, c_2, c_3, *, *, *, *)$, where c_l denotes constants, since the plaintexts are all of this form and there is no interaction between components. Since the four rightmost bytes take every possible value a slid pair occurs in a pool whenever the four leftmost bytes of X_i are (c_0, c_1, c_2, c_3). This happens with probability 2^{-32} for each plaintext under the assumption that $[\mathrm{MSM}]\,\mathrm{k}'_3\mathrm{Sk}_2\,[\mathrm{MSM}]\,\mathrm{k}'_1\mathrm{Sk}_0$ distributes values somewhat randomly. That is, the probability that a pool contains a slid pair is approximately $1 - e^{-1} \approx 0.63$.

3.4.5 Creating More Slid Pairs

Generating more slid pairs can be done in a number of different ways. One approach is using techniques similar to the Yoyo-game of [6].

If $P_0 = (p_0, p_1, \ldots, p_7)$, $C_0 = (c_0, c_1, \ldots, c_7)$ and $P_1 = (q_0, q_1, \ldots, q_7)$, $C_1 = (d_0, d_1, \ldots, d_7)$ is a slid pair, the texts $P'_0 = (q_0, p_1, \ldots, p_7)$, $C'_0 = \mathrm{E}(P'_0)$ and $C'_1 = (c_0, d_1, \ldots, d_7)$, $P'_1 = \mathrm{D}(C'_1)$ also form a slid pair. This follows from the lack of mixing between components and means that generating new slid pairs can be done by swapping components between the plaintexts, if the corresponding component is also swapped in the ciphertexts. By iterating this operation many slid pairs can be generated. Since the number of slid pairs is limited to 2^{64}, the process will eventually repeat, but the operation is a permutation on the set of slid pairs so it is easy to detect a cycle.

Writing the equations for the newly created slid pair

$$\mathrm{k}'_1\mathrm{Sk}_0\left(P'_0\right) = \mathrm{k}_4\mathrm{Sk}_5\left(C'_1\right)$$
$$\mathrm{k}'_1\mathrm{Sk}_0\left(P'_1\right) = \mathrm{k}_4\mathrm{Sk}_5\left(C'_0\right)$$

it is clear that the first equation, in P'_0 and C'_1, does not contribute any new information. The second equation, however, gives another 64-bit condition on the subkeys. That is, each additional slid pair created using this method adds one new 64-bit equation in the three subkeys.

Additional slid pairs can be generated by swapping several components at a time, yielding up to $2 \cdot 2^8$ slid pairs by swapping or not swapping each component. Obviously, swapping components where P_0 and P_1 are equal does not lead to new slid pairs.

3.4.6 Recovering the Key

Recovering the key can be done using the ideas discussed previously. There is enough information in one slid pair to recover the key, but we are not aware of a way of doing this with reasonable complexity. Using several slid pairs, the task of recovering the key is easier.

Using Equations 3.4 and 3.5 for several slid pairs created with the method of the previous section the key can be recovered. The simplest method is to guess the value of one byte of $\boldsymbol{k}_0$, $\boldsymbol{k}_5$ and $\boldsymbol{k}'_1 \oplus \boldsymbol{k}_4$ and use the slid pairs to check the guess. With three bytes of freedom for the subkey and a one byte condition from each equation, three equations would be needed to uniquely determine the subkey byte. However, since the first filtering of slid pairs leaves 2^{31} pairs, the complexity of this approach is higher than 2^{55} since we need to guess three bytes for each possible slid pair.

Instead of guessing bytes of the subkeys and checking the guesses, a table based approach can be used. The attacker precomputes a table with 2^{32} entries where the entries are suggested values for one byte of $\boldsymbol{k}_0$, $\boldsymbol{k}_5$ and $\boldsymbol{k}'_1 \oplus \boldsymbol{k}_4$, indexed on four bytes taken from one component of the two plaintexts and the two ciphertexts in a possible slid pair. For example, the table suggests values for $\boldsymbol{k}^0_0$, $\boldsymbol{k}^0_5$ and $\boldsymbol{k}'^0_1 \oplus \boldsymbol{k}^0_4$ when indexed by P^0_0, P^0_1, C^0_0 and C^0_1. Each pair suggests on the average 2^8 values of the key bytes. Using an additional pair, one suggestion is expected to remain.

The attack on 5 rounds of KHAZAD with a weak key, initially focusing on component $j = 4$, is as follows:

1. Request the encryption of 2^{32} plaintexts P_i of the form $(0, 0, 0, 0, *, *, *, *)$ giving ciphertexts C_i.

2. Sort the ciphertexts C_i on the four leftmost bytes. Consider all pairs where the ciphertexts are equal in the four leftmost bytes.

3. For each possible slid pair (P_0, C_0), (P_1, C_1) find the suggested values of component j of the key bytes using the lookup table.

4. Create two other possible slid pairs by swapping component j according to Section 3.4.5. Note that since all possible values in the last four components of the plaintexts were created in Step 1, only two chosen ciphertext queries are needed in this step. This step provides a 16-bit condition on the remaining suggested key values.

5. Repeat from Step 3 for component $j \in \{5, 6, 7\}$.

The complexity of the attack is as follows. Step 1 requires 2^{32} chosen plaintexts and after Step 2 the attacker is left with 2^{31} possible slid pairs since the sorting phase applies a 32-bit filter to 2^{63} pairs. Note that the sorting can be done for free by inserting the ciphertexts into a table indexed on the four leftmost bytes.

Step 3 gives approximately 2^8 suggestions for the three subkey bytes. Step 4 requires two chosen ciphertext queries for each remaining pair and gives a 16-bit condition on the key values, reducing the number of pairs to 2^{23}.

Each iteration done in Step 5 reduces the number of remaining pairs by a factor 2^8 and after three iterations 2^{-1} wrong pairs are expected to remain. During this filtering approximately $2 \cdot 2^{31} + 2 \cdot 2^{23} + 2 \cdot 2^{15} + 2 \cdot 2^7 < 2^{33}$ chosen ciphertexts are required.

The attack will succeed if the pool of 2^{32} plaintexts created in Step 1 contains a slid pair. The probability of a pool containing a slid pair is approximately $1 - e^{-1} \approx 0.63$ as shown previously.

The work required for the attack is less than 2^{40}, since at the first execution of Step 4 there are 2^{31} pairs and each suggests on the average 2^8 keys giving a total of 2^{39}, which is then reduced significantly in each step. The attack recovers 32 bits of the key and the remaining 32 unknown bits can be found, either by exhaustive search or by solving the equations in the remaining positions using the slid pairs created.

In summary, the attack requires 2^{32} chosen plaintexts, 2^{33} adaptively chosen ciphertexts, 2^{40} steps of computation and works for 2^{64} out of 2^{128} keys.

3.4.7 Other Ways

The sliding property of KHAZAD can be used in several other ways. For example, instead of swapping values as shown in Section 3.4.5, more slid pairs can be generated by making componentwise changes. That is, a small change in one component of P_0 and C_1 has probability better than 2^{-8} to also result in a slid pair.

Another idea, due to Christophe De Cannière, is to consider sets of plaintexts. If P_0, C_0 and P_1, C_1 is a slid pair, creating two sets of plaintexts and ciphertexts as follows results in two sets whose intermediate values during encryption and decryption collides.

Create the set $\mathcal{P}_0^*$ from P_0 by letting the first component take every possible value and the other components take the values found in P_0. That is, if $P_0 = (p_0, p_1, \ldots, p_7)$, create 256 plaintexts of the form $(x, p_1, \ldots, p_7)$, where x ranges over all possible values. Create $\mathcal{C}_1^*$ from C_1 in the same way. Due to the lack of interaction between components, encryption of the elements of the set $\mathcal{P}_0^*$ to the set $\mathcal{C}_0^*$ and decryption of $\mathcal{C}_1^*$ to $\mathcal{P}_1^*$, creates colliding sets. As the changes are in the same component in both $\mathcal{P}_0^*$ and $\mathcal{C}_1^*$ the sets will collide after the application of the initial $\mathrm{k}_1'\mathrm{Sk}_0$ and $\mathrm{k}_4\mathrm{Sk}_5$ respectively. Since the sets are the same, they will be the same after application of the involution $[\mathrm{MSM}]\,\mathrm{k}_2\mathrm{Sk}_3'\,[\mathrm{MSM}]$. The final $\mathrm{k}_5\mathrm{Sk}_4$ and $\mathrm{k}_0\mathrm{Sk}_1'$ respectively destroys this equivalence, but some properties survive. For example, the

distribution of the number of different values observed in each component survives and gives a strong filter for properly slid pairs.

This provides a membership test that requires 2^{39} chosen ciphertexts, which is worse than the key recovery attack given previously.

3.4.8 Extensions to More Rounds

Extending the attack to more rounds has not been successful. The main problem is that the componentwise nature of the equations is lost due to the introduction of M in the equations.

For 6 round KHAZAD, using the same slide as previously, the following equations can be written

$$\mathrm{k}_1'\mathrm{Sk}_0\left(P_0\right) = \mathrm{k}_4\mathrm{SMk}_5\mathrm{Sk}_6\left(C_1\right)$$
$$\mathrm{k}_1'\mathrm{Sk}_0\left(P_1\right) = \mathrm{k}_4\mathrm{SMk}_5\mathrm{Sk}_6\left(C_0\right)$$

The main difference compared to the equations for 5 rounds is the application of M. Since the branch number of M is 9, every component of the input influences every component of the output. The initial filtering of pairs, done by sorting in the attack on 5 rounds, is made impossible and solving for the key bytes can no longer be done for each component separately.

Naturally, the equations for 7 rounds of KHAZAD

$$\mathrm{k}_2'\mathrm{Sk}_1\mathrm{MSk}_0\left(P_0\right) = \mathrm{k}_5\mathrm{SMk}_6\mathrm{Sk}_7\left(C_1\right)$$
$$\mathrm{k}_2'\mathrm{Sk}_1\mathrm{MSk}_0\left(P_1\right) = \mathrm{k}_5\mathrm{SMk}_6\mathrm{Sk}_7\left(C_0\right)$$

are even more complex and hard to solve.

We conclude that the current attack is not a threat to the security of 8 round KHAZAD.

3.4.9 More Keys Using Differentials

The number of keys vulnerable to the basic attack as shown above is 2^{64}. The set of keys can be enlarged using ideas from differential cryptanalysis.

In Equations 3.2 and 3.3 we focus on the middle piece

$$\mathrm{k}_3'\mathrm{Sk}_2$$
$$\mathrm{k}_2\mathrm{Sk}_3'$$

It can be noted that the same values are xored both before and after the application of S. If $\boldsymbol{\Delta} = \boldsymbol{k}_2 \oplus \boldsymbol{k}_3'$ and $\boldsymbol{\Delta} = 0$ we have the case analyzed previously. If $\boldsymbol{\Delta} \neq 0$ the equality of values to the left can still be preserved if the difference $\boldsymbol{\Delta}$ introduced between the texts by the xor of $\boldsymbol{k}_2$ and $\boldsymbol{k}_3'$ is canceled by the xor of $\boldsymbol{k}_3'$ and $\boldsymbol{k}_2$ after application of S.

That is, we need differential characteristics of the form $\boldsymbol{\Delta} \rightarrow \boldsymbol{\Delta}$ through S. Of 255 nonzero differences, 135 has the required property through the $\mathrm{GF}(2^8) \rightarrow$

$GF(2^8)$ function s. The probabilities of these characteristics can be found by direct calculation and ranges from 2^{-7} to 2^{-5}.

For example, if $\boldsymbol{\Delta} = \boldsymbol{k}_2 \oplus \boldsymbol{k}'_3$ has one active component where the value is one of the 135 that can travel unaltered through S, the attack of Section 3.4.6 can be performed. Note however that swapping, as described in Section 3.4.5, no longer is guaranteed to result in another slid pair. Therefore, the key recovery phase can not reject pairs that give contradictions. Instead, counting the suggestions of key values can be used.

3.5 Permutation Cycle Structure of 7 Rounds

KHAZAD reduced to 7 rounds has a relatively simple description under weak keys similar to those discussed previously.

7 rounds of KHAZAD, can be written and rewritten as follows

$$\begin{aligned} &\mathrm{k}_7\mathrm{Sk}_6\mathrm{MSk}_5\mathrm{MSk}_4\mathrm{MSk}_3\mathrm{MSk}_2\mathrm{MSk}_1\mathrm{MSk}_0 = \\ &\mathrm{k}_7\mathrm{Sk}_6\mathrm{MSk}_5\,[\mathrm{MSM}]\,\mathrm{k}'_4\mathrm{Sk}_3\,[\mathrm{MSM}]\,\mathrm{k}'_2\mathrm{SMk}'_1\mathrm{Sk}_0 \end{aligned}$$

If $\mathrm{k}_3 = \mathrm{k}'_4$, the middle part $\mathrm{I} = [\mathrm{MSM}]\,\mathrm{k}'_4\mathrm{Sk}_3\,[\mathrm{MSM}]$ is an involution conjugate to S. $\mathrm{k}_5\mathrm{Ik}'_2$ follows Theorem 3.3 and since both involutions are fixed-point free Theorem 3.2 applies. Building further on this, $\mathrm{MSk}_5\mathrm{Ik}'_2\mathrm{SM}$, is conjugate to $\mathrm{k}_5\mathrm{Ik}'_2$ and therefore has the same cycle structure. $\mathrm{k}'_1\mathrm{Sk}_0$ and $\mathrm{k}_7\mathrm{Sk}_6$ are also compositions of two fixed-point free involutions by Theorem 3.3. This shows that 7 rounds of KHAZAD using a weak key can be written as the composition of three permutations, where each permutation follows Theorem 3.2.

We have not been able to devise an attack based on this observation.

3.6 Chapter Summary

In this chapter the block cipher KHAZAD was analyzed. KHAZAD is an 8 round iterated substitution permutation network with an involutional structure, a 64-bit block and a 128-bit key. It is based on a single fixed S-box, generated from two smaller involutions. The S-box itself is an involution. A linear layer, that is also an involution, based on a MDS code with byte level branch number 9 is used. The high rate of diffusion makes classical attacks such as differential and linear cryptanalysis hard to mount.

Some properties of KHAZAD, allowing us to rewrite KHAZAD into a form suitable for application of the slide with a twist attack, were described.

The best previous attack on KHAZAD was presented. It breaks 4 rounds and is claimed to require 2^9 chosen plaintexts and 2^{80} steps of computation. Some previous analysis regarding the permutation cycle structure of KHAZAD resulting from the involutional structure was also given.

The main result presented in the chapter was a set of 2^{64} weak keys in 5 round KHAZAD. The weak keys result from the involutional structure with the application

of a slide with a twist attack. A key in the set can be recovered using 2^{32} chosen plaintexts, 2^{33} adaptively chosen ciphertexts and 2^{40} work. The attack has a success probability approximately equal to 0.63.

Extensions to more rounds were considered but are hindered by the high rate of diffusion in the linear layer. Finally, it was noted that the set of weak keys can be enlarged using techniques from differential cryptanalysis.

Chapter 4

Analysis of SAFER++

This section contains our analysis of the 128-bit key version of the cipher SAFER++. After a brief history of the family of SAFER-ciphers a description of SAFER++ is given. This is followed by an overview of the analysis that has been done previously, both of the other ciphers in the SAFER-family and of SAFER++. We describe some interesting properties observed in the components of SAFER++ before describing our boomerang attack. The attack uses a novel middle round trick, exploiting the mathematical structure of the S-boxes, and can break up to 5.5 out of 7 rounds faster than exhaustive search. The results of this chapter are also in [10].

4.1 The Family of SAFER-ciphers

SAFER++ [38] is the latest cipher in the SAFER-family of ciphers. It was invented by Massey, Khachatrian and Kuregian and submitted to the European NESSIE project [50] in 2000.

The first cipher in the family, SAFER K-64 [36] with a 64-bit block, was introduced in 1993 by Massey and has since been extensively analyzed [26][28][30][42][44]. As a result of the analysis, Massey incorporated the revised key schedule suggested in [28] and called the resulting cipher SAFER SK. For the Advanced Encryption Standard [45] project, Massey, Khachatrian and Kuregian developed SAFER+ [37] with a 128-bit block, which was followed by SAFER++, submitted to the NESSIE project.

All the ciphers in the SAFER-family share the same S-boxes based on modular exponentiation. The ciphers are all based on a keyed nonlinear layer that is similar in all the ciphers, and a linear layer based on so called Pseudo Hadamard Transforms. The main difference between SAFER++ and the earlier ciphers is in the linear layer.

4.2 The Cipher SAFER++

The cipher SAFER++ was submitted to NESSIE in three different versions. The legacy version has a 64-bit block and a 128-bit key, while the normal version has a

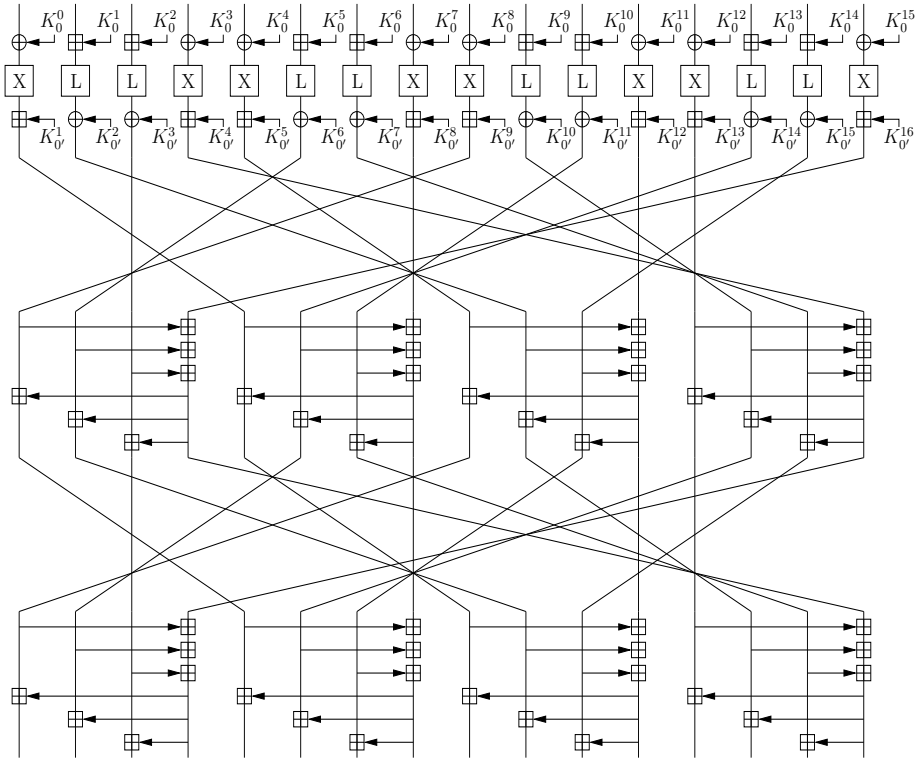

Figure 4.1. One round of SAFER++. X and L denotes the S-boxes based on modular exponentiation and discrete logarithms respectively, $\oplus$ denotes xor, $\boxplus$ denotes addition modulo 256 and K_i^j are the subkey bytes

128-bit block and a key of 128 or 256 bits. In this work, the focus is on the version with a 128-bit block and 128-bit key.

The inventors claim that SAFER++ "offers further substantial improvement over SAFER+" [38]. The main change is a 4-point Pseudo Hadamard Transform (PHT) instead of the 2-point PHT used in previous ciphers in the SAFER family. The inventors further claim that "all 5-round characteristics have probability significantly smaller than 2^{-128}" and that SAFER++ is secure against linear cryptanalysis after 2.5 rounds.

SAFER++ is an iterated product cipher in which every round consists of an upper key layer, a nonlinear substitution, a lower key layer and a linear transform. The upper key layer, the nonlinear substitution and the lower key layer together form the keyed nonlinear layer. The 128-bit block and 128-bit key version of SAFER++ studied here has 7 rounds, while the 128-bit block and 256-bit key version has 10 rounds. After the last round there is an output transformation, similar to the upper key layer.

One round of SAFER++ is depicted in Fig. 4.1. The 128-bit block is divided into 16 bytes and all operations are done byte-wise. Additions are done modulo 256 and we use subtraction modulo 256 as the difference operation.

4.2.1 The Keyed Nonlinear Layer

The keyed nonlinear layer performs a keyed substitution on the individual bytes of the block. The upper key layer combines a 16 byte subkey with the block. Bytes 0, 3, 4, 7, 8, 11, 12 and 15 of the subkey are xored to the corresponding bytes and bytes 1, 2, 5, 6, 9, 10, 13 and 14 are combined using addition modulo 256. A similar operation, the output transformation, is done after the last round.

The nonlinear layer uses two $\{0,1\}^8 \rightarrow \{0,1\}^8$ functions X and L defined as

$$\begin{aligned} \mathrm{X}(a) &= (45^a \bmod 257) \bmod 256 \\ \mathrm{L}(a) &= \log_{45}(a) \bmod 257 \end{aligned}$$

with the special case that $\mathrm{L}(0) = 128$, making L and X mutually inverse. The nonlinear layer applies X to bytes 0, 3, 4, 7, 8, 11, 12 and 15 and L to bytes 1, 2, 5, 6, 9, 10, 13 and 14.

The lower key layer combines the block with another 16 byte subkey. Bytes 0, 3, 4, 7, 8, 11, 12 and 15 are combined using addition modulo 256 and bytes 1, 2, 5, 6, 9, 10, 13 and 14 using xor.

4.2.2 The Linear Transform

The linear transform of SAFER++ is built from a so called 4-point Pseudo Hadamard Transform (4-PHT) and a coordinate permutation. The 4-PHT can be implemented with six modulo 256 additions and corresponds to multiplication by the matrix $\boldsymbol{H_4}$:

$$\boldsymbol{H_4} = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

That is, if the input to the 4-PHT is the row vector $\boldsymbol{x}$, the output is $\boldsymbol{x}\boldsymbol{H_4}$.

The linear layer first permutes the order of the bytes according to the permutation $[8, 5, 2, 15, 0, 13, 10, 7, 4, 1, 14, 11, 12, 9, 6, 3]$, meaning that byte 0 after the shuffle was byte 8 before the shuffle and so on. After the permutation, the 4-PHT is applied to groups of four consecutive bytes, that is, bytes 0, 1, 2 and 3 are combined, bytes 4, 5, 6 and 7 are combined and so on. After this first PHT layer, the same permutation of coordinates is done followed by another PHT layer.

In total, the linear layer can be seen as two applications of one linear function. This linear function can be represented by a matrix $\sqrt{\boldsymbol{A}}$ which is given in Appendix B together with the matrices $\boldsymbol{A}$ and $\boldsymbol{A}^{-1}$ corresponding to the linear layer and the inverse of the linear layer respectively.

4.2.3 The Key Schedule

The key schedule expands the 128- or 256-bit master key into the required number of subkeys. For the 128-bit key the cipher requires 15 subkeys, as there are 7 rounds each using two subkeys and an output transformation using one subkey.

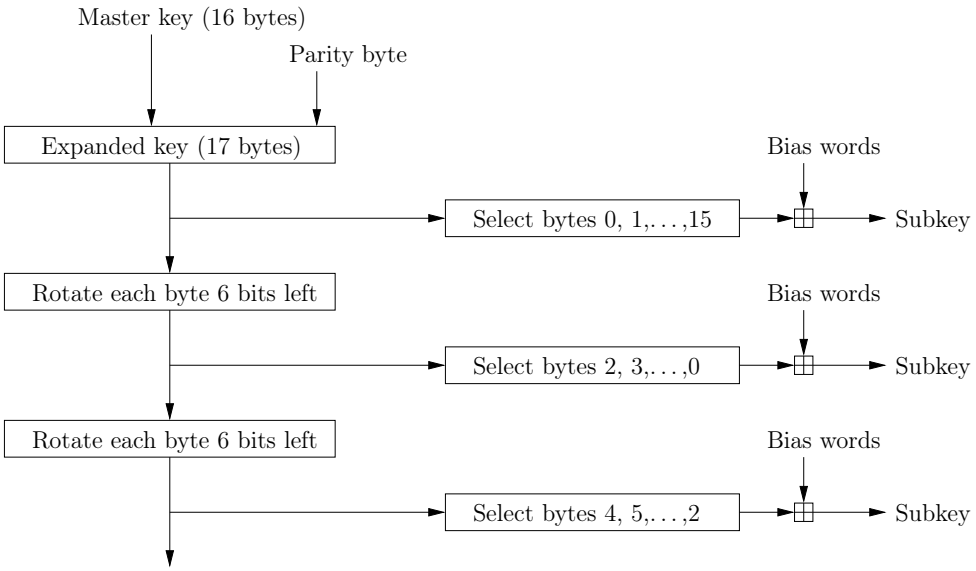

Figure 4.2. Generation of the subkeys for the upper key layer of the first, second and third round of SAFER++

In SAFER++ the key schedule consists of two parts. One part generates the subkeys for the upper key layer and the output transformation and the other part generates the subkeys for the lower key layer. The parts of the key schedule are very similar, differing only in the way the master key is loaded into the register. If the key is 128 bits it is expanded into 256 bits by concatenation with itself. The first 128 bits of the key are then used in the part generating the subkeys for the upper key layer and the last 128 bits of the key in generating the subkeys for the lower key layer.

The calculation of the subkeys uses a set of public constants, called bias words, and is done as follows. See Fig. 4.2 for an illustration of the generation of the subkeys for the upper key layer of the first three rounds.

1. Calculate a parity byte as the xor of the 16 key bytes and insert it as byte 16 in the register.

2. Select 16 consecutive bytes and output them as the subkey after addition of a bias word to each byte.

3. Rotate each of the 17 bytes of the register 6 bits to the left.

4. Repeat from Step 2, sliding the selection window two bytes, until all subkeys have been generated.

The other part of the subkey generation is done in the same way, except that each byte is rotated 3 bits to the left before Step 1 and that the sliding selection window of Step 2 starts at position 1 instead of 0. In view of the overlap in master

key bytes used in subkeys for consecutive rounds, the bytes of the subkeys will be numbered according to which master key byte they depend on and not according to their order in the subkey.

For example, the subkey used in the upper key layer of the first rounds is denoted by $(K_0^0, K_0^1, \ldots, K_0^{15})$ and the subkey used in the upper key layer of the second round is $(K_1^2, K_1^3, \ldots, K_1^0)$.

4.3 Previous Attacks

In this section we survey some of the analysis that has been done of previous ciphers in the SAFER-family as well as of SAFER++.

In [30] some results against SAFER K are summarized. Previously published in [31], an attack that breaks 5 out of 6 rounds with 2^{45} chosen plaintexts and 2^{37} time for the analysis is presented. The attack is a truncated differential attack that is much faster than exhaustive search of the key. These results were improved in [57] to breaking 6 rounds with 2^{53} chosen plaintexts and 2^{61} time.

Another weakness in SAFER K, previously published in [28], is in the key schedule. The weakness has the effect that for almost every key there exists on the average 3.5 other keys such that the encryption of plaintexts different in one byte only results in ciphertexts different in one byte only. This enables a related key attack using chosen plaintexts to recover the key, as well as finding collisions when using SAFER K in hashing modes.

In [42] the algebraic properties of the SAFER K linear transformation are investigated. It is shown that there exists invariant modules under the linear transformation and that this in connection with the key schedule leads to the existence of linear functions of the plaintext and ciphertext respectively that are dependent on only six out of eight key bytes. The revised key schedule suggested in [28] and later incorporated into SAFER SK ensures that the above mentioned dependence is on all key bytes.

The key schedule of the AES candidate SAFER+ was analyzed in [27]. A highly impractical meet-in-the-middle attack on SAFER+ with 256 bit keys requiring work approximately equal to 2^{240} encryptions shows that the key schedule originally used in SAFER+ is less than optimal. The key schedule was later tweaked to avoid the attack, and SAFER++ uses the tweaked key schedule.

4.3.1 Linear Cryptanalysis of SAFER++

The best previous attack on SAFER++ is linear cryptanalysis. The inventors of SAFER++ claim resistance against linear cryptanalysis after 2.5 rounds, but [43] shows that 3 rounds of SAFER++ with 128-bit keys is breakable with 2^{81} known plaintexts and 2^{101} steps for a fraction 2^{-13} of keys. For 256-bit keys the attack can break 3.5 rounds with 2^{81} known plaintexts and 2^{176} steps for a fraction 2^{-13} of keys. That is, there are weak key classes with respect to linear cryptanalysis for 3 round SAFER++ with 128-bit keys.

4.4 Interesting Properties

This section details some interesting properties observed in SAFER++. Some properties of the linear layer are investigated, followed by properties of the S-boxes and the key schedule.

4.4.1 Diffusion in the Linear Layer

The linear layer is based on a coordinate permutation and a 4-PHT as described previously. In [35] it is shown that the choice of coordinate permutation used in SAFER+ and SAFER++ provides "optimum transform diffusion", but the measure of diffusion used is not a particularly good one in our opinion. It measures the number of bytes that change in the output when one byte changes in the input but does not take into account changes in several bytes. Furthermore, the diffusion in the decryption direction is weaker with respect to one byte differences.

For example, in the encryption direction at least ten out of sixteen bytes change if one byte is changed, but in the decryption direction only five bytes are guaranteed to change when one byte is changed. A change in two bytes is only guaranteed to cause three changed bytes in the encryption direction. This means that the byte level branch number of the linear transform is less than or equal to 5.

After one encryption round, diffusion is complete in the sense that every byte can influence every other byte. The decryption direction is complete in this respect after two rounds.

As the linear transform is entirely based on addition modulo 256 and coordinate permutations the transformation inherits some properties of addition. For example, the most significant bit is special as it can only affect most significant bits due to the carry properties of addition. This makes differential characteristics with difference in the most significant bit only interesting. Since the bytes of SAFER++ are 8 bits wide, the most significant bit has value 128 or 80_x in hexadecimal notation.

4.4.2 Symmetry in the Linear Layer

Another interesting property observed in the linear layer is a four byte symmetry. We have not found an attack based on this property, but it is included here for future reference.

If the input to the linear layer is of the form

$$(a, b, c, d, a, b, c, d, a, b, c, d, a, b, c, d)$$

for some arbitrary values a, b, c and d the output will be of the form

$$(A, B, C, D, A, B, C, D, A, B, C, D, A, B, C, D)$$

for some values A, B, C and D. The nonlinear layer is symmetric in exactly the same way and were it not for the subkeys the property would hold for SAFER++ with an arbitrary number of rounds.

For example, two eigenvectors corresponding to the eigenvalue 1 of the linear transformation are

$$(0, 1, -1, 0, 0, 1, -1, 0, 0, 1, -1, 0, 0, 1, -1, 0)$$
$$(1, 0, -1, 0, 1, 0, -1, 0, 1, 0, -1, 0, 1, 0, -1, 0)$$

These vectors and all linear combinations of them are fixed points of the linear transformation.

4.4.3 Properties of the S-boxes

As the S-boxes of SAFER++ are based on mathematical functions we expect to find some interesting properties.

The following property, that was observed in [30] and independently rediscovered by us, holds for X and L

$$\mathrm{X}(a) + \mathrm{X}(a + 128) = (45^a \bmod 257) + (45^{a+128} \bmod 257) = 257 \equiv 1 \bmod 256$$

$$\mathrm{L}(a) - \mathrm{L}(1 - a) \equiv 128 \bmod 256$$

That is, if the inputs to X differ in the most significant bit only, the output values sum to 1. Similarly, if the sum of two values input to L is 1, the output will differ in the most significant bit only. This property will be used in our boomerang attack later in this chapter. Furthermore, since $a - (1 - a) = 2a - 1$, the output difference from X with inputs that differ in the most significant bit only is always odd.

4.4.4 Weak Key Schedule

The key schedule of SAFER++ has some properties worth mentioning. It can be noted that there is no interaction between key bytes in forming the subkeys, except for the parity byte. Since the bias words are known, a guess of a master key byte allows for the calculation of a subkey byte in almost all rounds.

In the 128-bit key version both registers of the key schedule are filled with the same key bytes, except for a byte-wise rotation, and the sliding window is one step out of phase making the subkeys of the upper and lower key layer of a round dependent to a large extent. This property is very useful when doing key guessing as it might be possible to learn several subkey bytes with only a few guesses of bytes of the master key.

It is in view of this property we choose to number the subkey bytes according to which master key byte they depend on and not according to their order in the subkey.

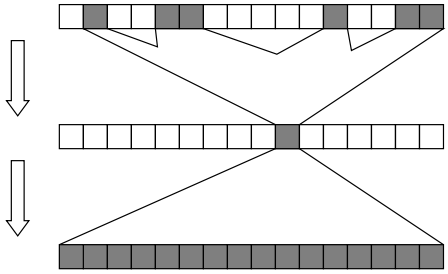

Figure 4.3. The truncated differential used in the top part of the boomerang distinguisher

4.5 A Boomerang Attack on SAFER++

4.5.1 A Boomerang Distinguisher for 4.5 Rounds

In this part we describe a boomerang distinguisher for 4.5 rounds of SAFER++. For an introduction to the boomerang attack, see Section 2.5.5. In the following, x denotes an arbitrary, nonzero value.

Denoting the linear layer by A and the keyed nonlinear layer by S, 4.5 rounds of SAFER++ can be written as [ASA − S − ASASA]. The first part, [ASA], will be covered top-down and the lower part, [ASASA], will be covered bottom-up. The middle part, [S], will be taken care of using the middle S-box trick. See Fig. 4.6 for an illustration of the boomerang used.

The top part of the boomerang uses a truncated differential over a round and a half, starting with difference $(0, x, 0, 0, x, x, 0, 0, 0, 0, 0, -4x, 0, 0, x, -x)$ that results in difference $(0, 0, 0, 0, 0, 0, 0, 0, 0, x, 0, 0, 0, 0, 0, 0)$ after the first linear layer. The output difference of the single active S-box is then diffused to up to 16 bytes in the next linear layer (see Fig. 4.3).

From the bottom, in the decryption direction, a truncated differential starting with difference $(0, 0, 128, 0, 128, 128, 0, 0, 0, 0, 0, 0, 128, 0, 0, 0)$ is used. This causes difference $(0, 0, 0, 0, 0, 128, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0)$ after the first linear layer and with probability 2^{-7} the difference after the keyed nonlinear layer is $(0, 0, 0, 0, 0, x, -x, 0, 0, 0, 0, 0, 0, 0, 0, 0)$. The probability is 2^{-7} since an input difference of 128 to X can not cause an even output difference. The two active bytes then cause bytes 3, 9, 11 and 14 to be active after the next linear layer. After one additional nonlinear and linear layer up to 16 bytes are active (see Fig. 4.4).

In a standard boomerang attack, the attacker would hope that the difference in the partial decryptions of the two pairs from the bottom are equal, making the difference on the last face of the boomerang equal to the difference on the first. This version of the attack is reasonably efficient against 4 rounds of SAFER++ and can be applied to any substitution permutation network. However, the probability of the event in the middle, where the parts of the boomerang meet, depends heavily on the number of active bytes and is often low. For the SAFER-family of ciphers we can do better by using special properties of the S-boxes.

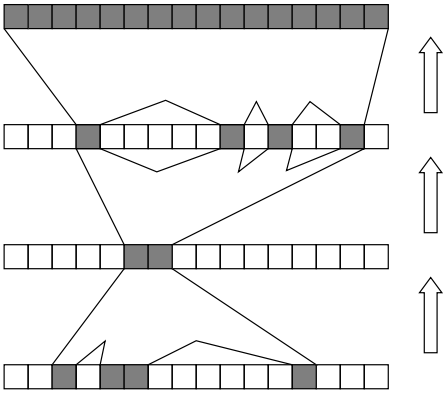

Figure 4.4. The truncated differential used in the bottom part of the boomerang distinguisher

4.5.2 The Middle Round S-box Trick

Using a property of the SAFER++ S-boxes, described in Section 4.4.3, we can make the boomerang cross the middle S-box layer for free. We use the fact that $\mathrm{X}(a) + \mathrm{X}(a + 128) \equiv 1 \bmod 256$, which holds with probability 1.

Consider the middle S-box layer where the top and bottom part of the boomerang meet (see Fig. 4.5). Suppose that the difference, coming from above, at the input to an X-box is 128. That is, values a and $a + 128$ are at the input. After the X-box the values will be of the form b and $1 - b$. Assume that on the other two faces we also have difference 128, coming from below. Then the fourth face of the boomerang has values $b + 128$ and $1 - b + 128$ on the lower side of the X-box. However, these values again sum to 1, since $128 + 128 \equiv 0 \bmod 256$. Therefore, the values at the top of the X-box will be of the form c and $c + 128$. Note that any pair of values summing to 0 modulo 256 would have worked instead of 128 and 128. The same reasoning holds for L since it is the inverse of X and the same difference is coming both from above and below in the boomerang.

This shows that if we have difference 128 coming from both the top and the bottom in the boomerang, the boomerang travels through the middle S-box layer for free due to the special properties of the S-boxes.

4.5.3 Breaking 5 and 5.5 Rounds

The distinguisher against 4.5 rounds of SAFER++ described above can be used to break 5 and 5.5 rounds. We first describe the attack on 5 rounds and then extend it to 5.5 rounds. See Fig. 4.6 for an illustration of the boomerang quartet used.

To the distinguisher described previously we add a keyed nonlinear layer at the top. Starting with a difference of the form $(0, *, 0, 0, *, *, 0, 0, 0, 0, 0, *, 0, 0, *, *)$, where $*$ denotes arbitrary differences, with probability 2^{-40} the required difference $(0, x, 0, 0, x, x, 0, 0, 0, 0, 0, -4x, 0, 0, x, -x)$ is created after the keyed nonlinear layer. This then propagates to one active S-box which with probability 2^{-8} gives output

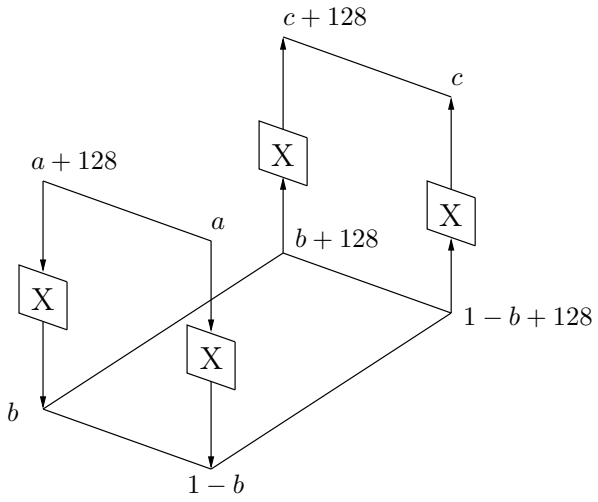

Figure 4.5. The free pass of the boomerang through the middle S-box layer

difference 128. The one-byte difference is then diffused to difference 128 in bytes 0, 1, 2, 3, 8, 9, 11, 13, 14 and 15 with no difference in the remaining bytes. In summary the upper part of the boomerang starts with differences in 6 bytes and with probability 2^{-48} the difference is 128 in 10 bytes after two rounds. This completes the upper part of the boomerang.

From the bottom the distinguisher starts with changes in the most significant bits of four bytes which gives two active bytes after one decryption round. With probability 2^{-7} the difference in the two active bytes is of the form x, $-x$ which propagates to four active S-boxes and further to difference 128 in all bytes except bytes 2, 4, 9 and 12 with probability $2^{-30.4}$. This probability was found by direct computation, using the fact that the input difference is odd and there are two X-boxes and two L-boxes active. In total, the probability of the lower part of the boomerang is $\left(2^{-7} \cdot 2^{-30.4}\right)^2$, since we need the differential to hold in two pairs.

Due to the middle S-box trick, the top part of the boomerang in the decryption direction propagates with probability 1. The total probability of the boomerang is thus $2^{-40} \cdot 2^{-8} \cdot \left(2^{-7} \cdot 2^{-30.4}\right)^2 = 2^{-122.8}$

The attack on 5 rounds is as follows:

1. Prepare a pool of 2^{48} plaintexts P_i, $i = 0, \ldots, 2^{48} - 1$ that have all possible values in bytes 1, 4, 5, 11, 14 and 15 and are constant in the other bytes. Encrypt the pool to get a pool of 2^{48} ciphertexts C_i.

2. Create a new pool of ciphertexts D_i from the pool C_i by changing the most significant bits in bytes $2, 4, 5$ and 12. Decrypt the pool D_i to obtain a pool Q_i of 2^{48} plaintexts.

3. Sort the pool Q_i on the bytes that correspond to the constant bytes in the pool P_i and pick the pairs Q_j, Q_k that have zero difference in those 10 bytes.

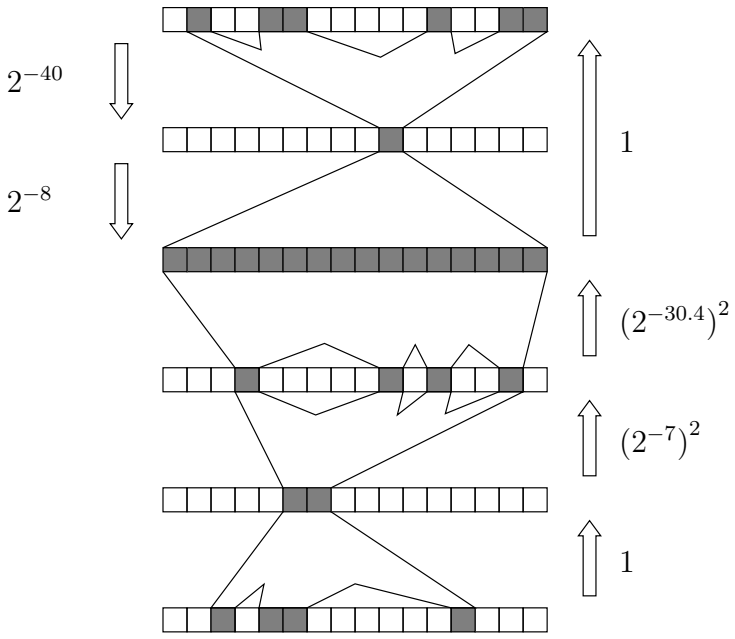

Figure 4.6. The boomerang quartet for SAFER++ reduced to 5.5 rounds

4. For each of the possibly good quartets P_j,P_k,Q_j,Q_k, guess the 3 key bytes K_0^4, K_0^{11} and K_0^{15}, do a partial encryption and check that the difference in the 3 bytes after the first nonlinear layer is of the form $(x, -4x, -x)$ with x odd, both in the P-pair and in the Q-pair (see Fig. 4.7). Note that we do not need to guess the key bytes added at the bottom of the keyed nonlinear layer, as they do not influence the subtractive difference. Keep the quartets that have the right difference, together with the key bytes that they suggest.

5. Guess the key byte K_0^{14}, do a partial encryption and check that the difference after xoring the key $K_{0'}^{15}$ (which is known from the previous step) at the bottom of the nonlinear layer is consistent with the difference found in Step 4. If no quartets survived, go to Step 1.

6. Guess the key bytes K_0^1 and $K_{0'}^2$, do a partial encryption and check that the difference after the first keyed nonlinear layer is the right one, consistent with the differences found in the earlier steps. Repeat this for the key bytes K_0^5 and $K_{0'}^6$.

7. Keep collecting suggestions for the 8 key bytes until one suggestion appears twice. For this suggestion, do an exhaustive search for the 8 remaining key bytes using trial encryption. If no consistent key is found, go to Step 1.

We now analyze the complexity of the attack. From the 2^{48} plaintexts created in Step 1 we get approximately 2^{95} pairs. Since the probability of the boomerang is $2^{-122.8}$, the probability is approximately $2^{-27.8}$ that a pool contains a boomerang.

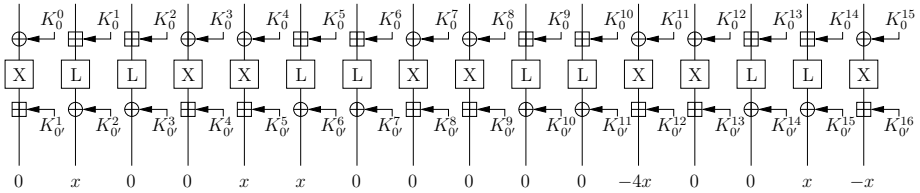

Figure 4.7. The subtractive differences in the top part of the boomerang where the key is recovered

After Step 3 we expect to have 2^{15} quartets left since we have an 80-bit condition on 2^{95} quartets. Step 4 leaves 2^5 quartets since there are three bytes to check in each pair, three bytes of freedom for the key and seven bits of freedom for the value of x in each pair leaving a ten bit condition in total. Similarly, after Step 5 about 2^{-3} quartets remain per pool. On the average, Step 6 will suggest one value for the four guessed key bytes per remaining quartet.

In order for the right key to be suggested twice in Step 7, two boomerangs are required. On the average, this will occur after analyzing 2^{28} pools. Using 2^{29} pools for an increased success probability, 2^{26} wrong keys will be suggested as well, but the chance that one of these wrong key values will be suggested twice is small. In fact, the probability is about $2^{51}/2^{64} = 2^{-13}$, implying that the exhaustive key search of Step 7 needs to be done for on average one suggestion only.

Since the probability of a probability p event occurring two times or more in N trials is $1 - \left((1-p)^N + p(1-p)^{N-1}\right)$ the probability of success of the attack can be calculated. Using $p = 2^{-27.8}$ and 2^{28}, 2^{29} and 2^{30} pools, the probability of success is approximately 0.68, 0.90 and 0.99 respectively.

With 2^{29} pools, where each pool contains 2^{48} chosen plaintexts and 2^{48} adaptively chosen ciphertexts, the data complexity of the attack is 2^{78}. Just collecting these texts will be the most computationally intensive step of the attack and the time complexity of the attack is therefore 2^{78}. Since each pool can be processed independently, the memory requirement of the attack is 2^{48} blocks and some small amount of storage for the suggested key values.

The attack on 5 rounds can be extended to 5.5 rounds by guessing 30 bits of the key in positions 2, 4, 5 and 12 at the end of the added keyed nonlinear layer. The data and time complexity of the 5.5 round attack is thus 2^{108}. Note that two key bits are saved by using the special properties of the L-box.

4.5.4 Improving the Results

Extending the attack to more rounds turns out to be difficult. One approach would be to search for longer truncated differentials. We have not been successful using this approach as the probabilities of the longer differentials investigated are to low to allow an attack.

Another approach, that has not been investigated by us, is the use of amplified boomerang techniques [25]. Such techniques can sometimes allow for boomerang

attacks to be carried out using chosen plaintext only, but would still need longer truncated differentials to break more rounds.

4.6 Chapter Summary

In this chapter the block cipher SAFER++ was analyzed. After some brief history concerning the previous ciphers in the SAFER-family a description of SAFER++ was given. SAFER++ is a 7 round iterated substitution permutation network, built from a keyed nonlinear layer based on modular addition, xor and modular exponentiation, and a linear transformation based on a Pseudo Hadamard Transform and a coordinate permutation.

Several interesting properties of the components of SAFER++ were presented, for example a differential-summation property that was used later in the chapter to improve a boomerang attack.

Some attacks on previous ciphers in the SAFER-family were summarized together with the best attack on SAFER++. Previous to the work presented in this chapter, the best published attack on SAFER++ was linear cryptanalysis, breaking three rounds of the 128-bit key version with 2^{81} known plaintexts and 2^{101} steps of analysis for a fraction 2^{-13} of keys.

The main result of the chapter was a boomerang attack that can be used to break up to 5.5 out of 7 rounds of SAFER++. The attack uses a combination of differential characteristics, truncated differentials and a novel middle round trick, based on the differential-summation property described earlier in the chapter. The complexity of the attack on 5 rounds is 2^{78} chosen plaintexts and adaptively chosen ciphertexts with similar time complexity and a success probability better than 0.90. Extending the attack to 5.5 rounds can be done by guessing 30 key bits, bringing the complexity up to 2^{108} data and work.

The attack given, while not directly applicable to the full SAFER++ cipher, shows that the margin of security is less than or equal to a round and a half. This may not inspire the trust required and therefore users of block ciphers might want to consider other alternatives than SAFER++. For example, the NESSIE project choose to not include SAFER++ in its portfolio of recommended cryptographic primitives "because of some concerns, both about certain structural properties of SAFER++$_{128}$ and about the low security margin" [49].

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Our analysis has been focused on two recently proposed ciphers, KHAZAD and SAFER++.

The analysis of KHAZAD does not threaten the full cipher, as our best result is the existence of weak keys in a version reduced to 5 from 8 rounds. The simple mathematical structure that results in the weak keys we investigate might also have other implications. We are not aware of any results against more rounds, but due to the simplicity of the cipher and the relatively limited amount of analysis performed of it we do not recommend it be used in situations where security is important. However, as a target of further analysis KHAZAD is interesting.

Our attack on SAFER++ breaks 5.5 out of 7 rounds faster than exhaustive search for 128 bit keys. The attack is not practical as the amount of data and time required for the attack is high. However, the attack is much faster than exhaustive search and in our opinion it shows that the security margin of SAFER++ is too small. We do not recommend the use of SAFER++ because of these concerns.

It can be noted that the final decision [49] of the NESSIE project, reached after the publication of a draft paper containing the results of Chapter 4, was to not recommend the use of KHAZAD or SAFER++.

5.2 Future Work

In this section we discuss some ideas for future work with respect to the results of this thesis.

The results of this thesis do not apply to the full ciphers. Our results are restricted to round-reduced versions and naturally, extending the results to more rounds would be interesting.

For KHAZAD extending the weak keys to more rounds might be possible, perhaps by using the relations between the subkeys resulting from the weak key restrictions. General attacks, not relying on weak keys, resulting from the involutional structure

is another open problem. Since the diffusion of the linear layer is strong and the nonlinearity of the substitution step is high classical attacks such as linear and differential cryptanalysis seem hard to mount, suggesting that research possibly should be focused on other properties. The permutation cycle structure, as described in Section 3.3.2, together with the observations of Section 3.5 is perhaps one such property.

Concerning the attack on SAFER++, some improvements might be possible for example by considering the use of amplified boomerang techniques [25]. Breaking more rounds has not been successful, but might be possible by using better differentials. Application of our techniques to the previous ciphers in the SAFER-family could also be worthwhile.

Another open problem is the application of our boomerang techniques to other substitution permutation networks. Apart from the middle round S-box trick, the techniques are applicable to all substitution permutation networks with incomplete diffusion. For example, 4 and 5 rounds of RIJNDAEL can be broken with a similar attack. Tricks similar to our S-box trick can probably be used against other ciphers where the S-box has a simple mathematical structure.

References

[1] Ross J. Anderson. *Security Engineering - A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, 2001.

[2] Paulo S. L. M. Barreto and Vincent Rijmen. *The Anubis Block Cipher*. Primitive submitted to NESSIE, September 2000.

[3] Paulo S. L. M. Barreto and Vincent Rijmen. *The Khazad Legacy-Level Block Cipher*. Primitive submitted to NESSIE, September 2000.

[4] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 1996.

[5] Eli Biham. New types of cryptanalytic attacks using related keys. In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 398–409. Springer-Verlag, 1993.

[6] Eli Biham, Alex Biryukov, Orr Dunkelman, Eran Richardson, and Adi Shamir. Initial observations on Skipjack: Cryptanalysis of Skipjack-3XOR. In Stafford Tavares and Henk Meijer, editors, *Selected Areas in Cryptography, SAC 1998*, volume 1556 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, 1998.

[7] Eli Biham, Orr Dunkelman, and Nathan Keller. The rectangle attack – rectangling the Serpent. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 340–357. Springer-Verlag, 2001.

[8] Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round DES. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. Springer-Verlag, 1992.

[9] Eli Biham and Adi Shamir. *Differential cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993.

[10] Alex Biryukov, Christophe De Cannière, and Gustaf Dellkrantz. Cryptanalysis of SAFER++. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*,

volume 2729 of *Lecture Notes in Computer Science*, pages 195–211. Springer-Verlag, 2003.

[11] Alex Biryukov, Jorge Nakahara, Jr, Bart Preneel, and Joos Vandewalle. New weak-key classes of IDEA. In Robert H. Deng, Sihan Qing, Feng Bao, and Jianying Zhou, editors, *Proceedings of ICICS'02*, volume 2513 of *Lecture Notes in Computer Science*, pages 315–326. Springer-Verlag, 2002. Also in *Proceedings of the Third NESSIE Workshop*, 2002.

[12] Alex Biryukov and Bart Preneel. Preliminary analysis of Khazad. In *Proceedings of the Third NESSIE Workshop*. NESSIE, November 2002.

[13] Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 394–405. Springer-Verlag, 2001.

[14] Alex Biryukov and David Wagner. Slide attacks. In Lars R. Knudsen, editor, *Fast Software Encryption, FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer-Verlag, 1999.

[15] Alex Biryukov and David Wagner. Advanced slide attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 589–606. Springer-Verlag, 2000.

[16] Joan Daemen. *Cipher and Hash Function Design Strategies Based on Linear and Differential Cryptanalysis*. Doctoral dissertation, K. U. Leuven, March 1995.

[17] Joan Daemen, René Govaerts, and Joos Vandewalle. Weak keys for IDEA. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 224–231. Springer-Verlag, 1994.

[18] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *Fast Software Encryption, FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer-Verlag, 1997.

[19] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — The Advanced Encryption Standard*. Springer-Verlag, 2002.

[20] Horst Feistel. *Block Cipher Cryptographic System*. U.S. Patent #3,798,359, 19 March 1974.

[21] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography, SAC 2001*, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag, 2001.

[22] Henri Gilbert and Marine Minier. A collision attack on seven rounds of Rijndael. In *Proceedings of the Third AES Candidate Conference*, pages 230–241. National Institute of Standards and Technology, April 2000.

[23] Philip Hawkes. Differential-linear weak key classes of IDEA. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 112–126. Springer-Verlag, 1998.

[24] Thomas Jakobsen and Lars R. Knudsen. The interpolation attack on block ciphers. In Eli Biham, editor, *Fast Software Encryption, FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 28–40. Springer-Verlag, 1997.

[25] John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified boomerang attacks against reduced-round MARS and Serpent. In Bruce Schneier, editor, *Fast Software Encryption, FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 75–93. Springer-Verlag, 2001.

[26] John Kelsey, Bruce Schneier, and David Wagner. Key-schedule cryptanalysis of 3-WAY, IDEA, G-DES, RC4, SAFER, and Triple-DES. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 237–251. Springer-Verlag, 1996.

[27] John Kelsey, Bruce Schneier, and David Wagner. Key schedule weaknesses in SAFER+. In *Proceedings of the Second AES Candidate Conference*, pages 155–167. National Institute of Standards and Technology, March 1999.

[28] Lars R. Knudsen. A key-schedule weakness in SAFER K-64. In Don Coppersmith, editor, *Advances in Cryptology – CRYPTO'95*, volume 963 of *Lecture Notes in Computer Science*, pages 274–286. Springer-Verlag, 1995.

[29] Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *Fast Software Encryption, FSE'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer-Verlag, 1995.

[30] Lars R. Knudsen. A detailed analysis of SAFER K. *Journal of Cryptology*, 13(4):417–436, 2000.

[31] Lars R. Knudsen and Thomas A. Berson. Truncated differentials of SAFER. In Dieter Gollmann, editor, *Fast Software Encryption, FSE'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 15–26. Springer-Verlag, 1996.

[32] Xuejia Lai and James L. Massey. A proposal for a new block encryption standard. In Ivan B. Damgård, editor, *Advances in Cryptology – EUROCRYPT'90*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer-Verlag, 1990.

[33] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In Donald W. Davies, editor, *Advances in Cryptology – EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer-Verlag, 1991.

[34] Stefan Lucks. The saturation attack - a bait for Twofish. In Mitsuru Matsui, editor, *Fast Software Encryption, FSE 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2002.

[35] James Massey. On the optimality of SAFER+ diffusion. In *Proceedings of the Second AES Candidate Conference*. National Institute of Standards and Technology, March 1999.

[36] James L. Massey. SAFER K-64: A byte-oriented block-ciphering algorithm. In Ross J. Anderson, editor, *Fast Software Encryption, FSE'93*, volume 809 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, 1994.

[37] James L. Massey, Gurgen H. Khachatrian, and Melsik K. Kuregian. *Nomination of SAFER+ as Candidate Algorithm for the Advanced Encryption Standard (AES)*. Primitive submitted to NIST by Cylink Corp., June 1998.

[38] James L. Massey, Gurgen H. Khachatrian, and Melsik K. Kuregian. *Nomination of SAFER++ as Candidate Algorithm for the New European Schemes for Signatures, Integrity, and Encryption (NESSIE)*. Primitive submitted to NESSIE by Cylink Corp., September 2000.

[39] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer-Verlag, 1993.

[40] Mitsuru Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1994.

[41] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997. Available at http://www.cacr.math.uwaterloo.ca/hac/.

[42] Sean Murphy. An analysis of SAFER. *Journal of Cryptology*, 11(4):235–251, 1998.

[43] Jorge Nakahara Jr. *An Update to Linear Cryptanalysis of SAFER++*. Internal report NES/DOC/KUL/WP5/031/1, NESSIE, 2003.

[44] Jorge Nakahara Jr, Bart Preneel, and Joos Vandewalle. Linear cryptanalysis of reduced-round versions of the SAFER block cipher family. In Bruce Schneier, editor, *Fast Software Encryption, FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 244–261. Springer-Verlag, 2001.

[45] National Institute of Standards and Technology. *Advanced Encryption Standard (AES)*. http://csrc.nist.gov/CryptoToolkit/aes/.

[46] National Institute of Standards and Technology. *Data Encryption Standard (DES)*. FIPS-46-3. NIST, October 1999. Available at http://csrc.nist.gov/publications/fips/.

[47] National Institute of Standards and Technology. *Recommendation for Block Cipher Modes of Operation*. SP-800-38A. NIST, December 2001. Available at http://csrc.nist.gov/encryption/.

[48] National Institute of Standards and Technology. *The Keyed-Hash Message Authentication Code (HMAC)*. FIPS-198. NIST, March 2002. Available at http://csrc.nist.gov/publications/fips/, based on [4].

[49] NESSIE Consortium. *Portfolio of recommended cryptographic primitives*. Final decision, NESSIE, February 2003.

[50] NESSIE Project – New European Schemes for Signatures, Integrity and Encryption, IST-1999-12324. http://cryptonessie.org.

[51] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 55–64. Springer-Verlag, 1993.

[52] Marian Rejewski. An application of the theory of permutations in breaking the Enigma cipher. *Applicaciones Mathematicae*, 16(4), Warsaw, 1980. Version transcribed by Enrico Grigolon available at http://frode.home.cern.ch/frode/crypto/index.html.

[53] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). In Ross J. Anderson, editor, *Fast Software Encryption, FSE'93*, volume 809 of *Lecture Notes in Computer Science*, pages 191–204. Springer-Verlag, 1994.

[54] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 2nd edition, 1996.

[55] Claude E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[56] David Wagner. The boomerang attack. In Lars R. Knudsen, editor, *Fast Software Encryption, FSE'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer-Verlag, 1999.

[57] Hongjun Wu, Feng Bao, Robert H. Deng, and Qin-Zhong Ye. Improved truncated differential attacks on SAFER. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology – ASIACRYPT'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 133–147. Springer-Verlag, 1998.

Appendix A

The Nonlinear Functions of Khazad

This appendix contains the tables for functions P, Q and s used in the nonlinear layer of the block cipher Khazad.

a	$\mathrm{P}(a)$	$\mathrm{Q}(a)$
0_x	3_x	9_x
1_x	$\mathtt{F}_x$	$\mathtt{E}_x$
2_x	$\mathtt{E}_x$	5_x
3_x	0_x	6_x
4_x	5_x	$\mathtt{A}_x$
5_x	4_x	2_x
6_x	$\mathtt{B}_x$	3_x
7_x	$\mathtt{C}_x$	$\mathtt{C}_x$
8_x	$\mathtt{D}_x$	$\mathtt{F}_x$
9_x	$\mathtt{A}_x$	0_x
$\mathtt{A}_x$	9_x	4_x
$\mathtt{B}_x$	6_x	$\mathtt{D}_x$
$\mathtt{C}_x$	7_x	7_x
$\mathtt{D}_x$	8_x	$\mathtt{B}_x$
$\mathtt{E}_x$	2_x	1_x
$\mathtt{F}_x$	1_x	8_x

s	00_x	01_x	02_x	03_x	04_x	05_x	06_x	07_x	08_x	09_x	$0A_x$	$0B_x$	$0C_x$	$0D_x$	$0E_x$	$0F_x$
00_x	BA_x	54_x	$2F_x$	74_x	53_x	$D3_x$	$D2_x$	$4D_x$	50_x	AC_x	$8D_x$	BF_x	70_x	52_x	$9A_x$	$4C_x$
10_x	EA_x	$D5_x$	97_x	$D1_x$	33_x	51_x	$5B_x$	$A6_x$	DE_x	48_x	$A8_x$	99_x	DB_x	32_x	$B7_x$	FC_x
20_x	$E3_x$	$9E_x$	91_x	$9B_x$	$E2_x$	BB_x	41_x	$6E_x$	$A5_x$	CB_x	$6B_x$	95_x	$A1_x$	$F3_x$	$B1_x$	02_x
30_x	CC_x	$C4_x$	$1D_x$	14_x	$C3_x$	63_x	DA_x	$5D_x$	$5F_x$	DC_x	$7D_x$	CD_x	$7F_x$	$5A_x$	$6C_x$	$5C_x$
40_x	$F7_x$	26_x	FF_x	ED_x	$E8_x$	$9D_x$	$6F_x$	$8E_x$	19_x	$A0_x$	$F0_x$	89_x	$0F_x$	07_x	AF_x	FB_x
50_x	08_x	15_x	$0D_x$	04_x	01_x	64_x	DF_x	76_x	79_x	DD_x	$3D_x$	16_x	$3F_x$	37_x	$6D_x$	38_x
60_x	$B9_x$	73_x	$E9_x$	35_x	55_x	71_x	$7B_x$	$8C_x$	72_x	88_x	$F6_x$	$2A_x$	$3E_x$	$5E_x$	27_x	46_x
70_x	$0C_x$	65_x	68_x	61_x	03_x	$C1_x$	57_x	$D6_x$	$D9_x$	58_x	$D8_x$	66_x	$D7_x$	$3A_x$	$C8_x$	$3C_x$
80_x	FA_x	96_x	$A7_x$	98_x	EC_x	$B8_x$	$C7_x$	AE_x	69_x	$4B_x$	AB_x	$A9_x$	67_x	$0A_x$	47_x	$F2_x$
90_x	$B5_x$	22_x	$E5_x$	EE_x	BE_x	$2B_x$	81_x	12_x	83_x	$1B_x$	$0E_x$	23_x	$F5_x$	45_x	21_x	CE_x
$A0_x$	49_x	$2C_x$	$F9_x$	$E6_x$	$B6_x$	28_x	17_x	82_x	$1A_x$	$8B_x$	FE_x	$8A_x$	09_x	$C9_x$	87_x	$4E_x$
$B0_x$	$E1_x$	$2E_x$	$E4_x$	$E0_x$	EB_x	90_x	$A4_x$	$1E_x$	85_x	60_x	00_x	25_x	$F4_x$	$F1_x$	94_x	$0B_x$
$C0_x$	$E7_x$	75_x	EF_x	34_x	31_x	$D4_x$	$D0_x$	86_x	$7E_x$	AD_x	FD_x	29_x	30_x	$3B_x$	$9F_x$	$F8_x$
$D0_x$	$C6_x$	13_x	06_x	05_x	$C5_x$	11_x	77_x	$7C_x$	$7A_x$	78_x	36_x	$1C_x$	39_x	59_x	18_x	56_x
$E0_x$	$B3_x$	$B0_x$	24_x	20_x	$B2_x$	92_x	$A3_x$	$C0_x$	44_x	62_x	10_x	$B4_x$	84_x	43_x	93_x	$C2_x$
$F0_x$	$4A_x$	BD_x	$8F_x$	$2D_x$	BC_x	$9C_x$	$6A_x$	40_x	CF_x	$A2_x$	80_x	$4F_x$	$1F_x$	CA_x	AA_x	42_x

Appendix B

The Linear Layer of SAFER++

This appendix contains the matrices corresponding to half of the linear layer, the linear layer and the inverse of the linear layer of SAFER++.

$$\sqrt{\boldsymbol{A}} = \begin{pmatrix}
0 & 0 & 0 & 0 & 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 2 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

$$\boldsymbol{A} = \begin{pmatrix}
1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 2 & 2 & 2 & 1 & 1 & 2 & 1 \\
2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 4 & 2 & 2 \\
2 & 2 & 4 & 2 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 2 & 1 & 1 & 1 \\
4 & 2 & 2 & 2 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\
1 & 1 & 2 & 1 & 2 & 1 & 1 & 1 & 2 & 4 & 2 & 2 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 2 & 2 & 4 & 2 & 2 & 1 & 1 & 1 \\
1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 \\
1 & 1 & 2 & 1 & 4 & 2 & 2 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 2 & 4 & 2 & 2 & 1 & 1 & 2 & 1 & 2 & 1 & 1 & 1 \\
1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 2 & 2 & 4 & 2 \\
2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 4 & 2 & 2 & 2 \\
2 & 4 & 2 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 \\
2 & 1 & 1 & 1 & 2 & 2 & 4 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 \\
1 & 1 & 2 & 1 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}$$

$$\boldsymbol{A}^{-1} = \begin{pmatrix}
0 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -4 & 0 & 0 & 1 & -1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & -1 \\
0 & 0 & -1 & 16 & -1 & 0 & -4 & 1 & 0 & -4 & 0 & 1 & -4 & -1 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & -1 \\
1 & 0 & 0 & -1 & 0 & 0 & 0 & -4 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 1 & -1 & 0 & 1 & 1 & 0 \\
-4 & 0 & 0 & 1 & 0 & 0 & 0 & 16 & -1 & -1 & -4 & 1 & 0 & -4 & -1 & 1 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & -4 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 1 & -1 \\
0 & 1 & 0 & -1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & -4 & 0 & 0 & 1 & 0 \\
-1 & -4 & 0 & 1 & -4 & -1 & -1 & 1 & 0 & 0 & 0 & 16 & 0 & 0 & -4 & 1 \\
0 & 0 & 1 & -1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & -4 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & -4 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & -4 \\
0 & -1 & -4 & 1 & 0 & -4 & 0 & 1 & -4 & 0 & -1 & 1 & -1 & 0 & 0 & 16
\end{pmatrix}$$